

Real-time Software Hand Pose Recognition using Single View Depth Images

Supervisor: Dr Herman Arnold Engelbrecht

by

Stefan Francois Alberts

Thesis presented in fulfilment of the requirements for the degree
Master of Engineering (Research) in the Faculty of Engineering
at Stellenbosch



April 2014

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

April 2014

Abstract

The fairly recent introduction of low-cost depth sensors such as Microsoft's Xbox Kinect has encouraged a large amount of research on the use of depth sensors for many common Computer Vision problems. Depth images are advantageous over normal colour images because of how easily objects in a scene can be segregated in real-time. Microsoft used the depth images from the Kinect to successfully separate multiple users and track various larger body joints, but has difficulty tracking smaller joints such as those of the fingers. This is a result of the low resolution and noisy nature of the depth images produced by the Kinect.

The objective of this project is to use the depth images produced by the Kinect to remotely track the user's hands and to recognise the static hand poses in real-time. Such a system would make it possible to control an electronic device from a distance without the use of a remote control. It can be used to control computer systems during computer aided presentations, translate sign language and to provide more hygienic control devices in clean rooms such as operating theatres and electronic laboratories.

The proposed system uses the open-source OpenNI framework to retrieve the depth images from the Kinect and to track the user's hands. Random Decision Forests are trained using computer generated depth images of various hand poses and used to classify the hand regions from a depth image. The region images are processed using a Mean-Shift based joint estimator to find the 3D joint coordinates. These coordinates are finally used to classify the static hand pose using a Support Vector Machine trained using the libSVM library. The system achieves a final accuracy of 95.61% when tested against synthetic data and 81.35% when tested against real world data.

Opsomming

Die onlangse bekendstelling van lae-koste diepte sensors soos Microsoft se Xbox Kinect het groot belangstelling opgewek in navorsing oor die gebruik van die diepte sensors vir algemene Rekenaarvisie probleme. Diepte beelde maak dit baie eenvoudig om intyds verskillende voorwerpe in 'n toneel van mekaar te skei. Microsoft het diepte beelde van die Kinect gebruik om verskeie persone en hul ledemate suksesvol te volg. Dit kan egter nie kleiner ledemate soos die vingers volg nie as gevolg van die lae resoluusie en voorkoms van geraas in die beelde.

Die doel van hierdie projek is om die diepte beelde (verkry vanaf die Kinect) te gebruik om intyds 'n gebruiker se hande te volg oor 'n afstand en die statiese handgebare te herken. So 'n stelsel sal dit moontlik maak om elektroniese toestelle oor 'n afstand te kan beheer sonder die gebruik van 'n afstandbeheerder. Dit kan gebruik word om rekenaarstelsels te beheer gedurende rekenaargesteunde aanbiedings, vir die vertaling van vingertaal en kan ook gebruik word as higiëniese, tasvrye beheer toestelle in skoonkamers soos operasieteatres en elektroniese laboratoriums.

Die voorgestelde stelsel maak gebruik van die oopbron OpenNI raamwerk om die diepte beelde vanaf die Kinect te lees en die gebruiker se hande te volg. Lukrake Besluitnemingswoude ("Random Decision Forests") is opgelei met behulp van rekenaar gegenereerde diepte beelde van verskeie handgebare en word gebruik om die verskeie handdele vanaf 'n diepte beeld te klassifiseer. Die 3D koördinate van die hand ledemate word dan verkry deur gebruik te maak van 'n Gemiddelde-Afset gebaseerde ledemaat herkenner. Hierdie koördinate word dan gebruik om die statiese handgebaar te klassifiseer met behulp van 'n Steun-Vektor Masjien ("Support Vector Machine"), opgelei met behulp van die libSVM biblioteek. Die stelsel behaal 'n finale akkuraatheid van 95.61% wanneer dit getoets word teen sintetiese data en 81.35% wanneer getoets word teen werklike data.

Acknowledgements

First and foremost, I would like to acknowledge God. Without His grace I would not have been able to complete this thesis.

I would like to thank my parents, Nico and Beverly Alberts. Your love and support have always driven me to give my best.

I would like to thank my supervisor, Dr Herman Engelbrecht. His guidance helped me to deliver a product of higher quality by proof reading various drafts and giving general advice during the development of the practical system.

Thanks to my bursary company, MIH, for providing the funding necessary to complete my Master's Degree.

Lastly, thanks to everyone who helped me with the testing of the system, especially my fellow students in the Media Lab. You all made it a pleasure to work in the lab.

Contents

Declaration	i
Abstract	ii
Opsomming	iii
Acknowledgements	iv
Contents	v
List of Figures	x
List of Tables	xv
Nomenclature	xviii
1 Introduction	1
1.1 Gesture Recognition System Applications	1
1.2 Existing Systems	2
1.2.1 GestureTek	2
1.2.2 Microsoft Kinect	2
1.3 Research Problem	4
1.4 Gesture Recognition Problem	4
1.4.1 Hand Tracking	7
1.4.2 Pose Recognition	8
1.5 Objectives	9
1.6 Contributions	10
1.7 Thesis Overview	11
2 System Overview	13
2.1 Depth Sensor Interface	13
2.1.1 OpenNI	13
2.1.2 Data Processing	14

CONTENTS

vi

2.1.3	Camera Calibration	15
2.2	Hand Tracking	16
2.2.1	OpenNI NITE	16
2.2.2	Skeltrack	17
2.2.3	Implementation	18
2.3	Pose Recognition	18
2.3.1	Pose Recognition Problem and Solution	19
2.3.2	Proposed System	22
2.3.3	Recognised Poses	23
2.3.4	Input Data Constraints	25
2.4	Summary	27
3	Decision Trees	28
3.1	Definition	28
3.2	Training	30
3.2.1	Training Procedure Overview	30
3.2.2	Split Types	31
3.2.3	Split Criterion	33
3.2.4	Classification Rule	35
3.2.5	Stop Rule	36
3.3	Decision Forests	38
3.4	Random Decision Trees	40
3.5	Summary	41
4	Region Classifier	43
4.1	Generating Synthetic Images	43
4.1.1	Rendering Application	44
4.1.2	3D Hand Model	44
4.1.3	Image Generation	45
4.1.4	Generation of Training and Test Sets	46
4.2	Trainer Design	48
4.2.1	Depth Features	48
4.2.2	Random Split Generation	49
4.2.3	Speed	50
4.2.4	Memory Management	51
4.2.5	Security	53
4.3	Testing	54
4.3.1	Experimental Setup	54
4.3.2	Accuracy of Classifiers	55
4.3.3	Speed of Classifiers	61

4.3.4	Confusion Properties	62
4.4	Summary	66
5	Joint Position Estimation	67
5.1	Centre of Gravity Joint Estimation	67
5.2	Mean-Shift Algorithm	68
5.2.1	Density Estimator	68
5.2.2	Density Gradient Estimator	69
5.2.3	Introduction of the Mean-Shift	69
5.3	System Implementation	71
5.3.1	Calculating the Mean-Shift	71
5.3.2	Estimating the Hand Joints	71
5.3.3	Joint Reservation Algorithm	72
5.3.4	Final Joint Coordinates	78
5.4	Testing	78
5.4.1	Experimental Setup	79
5.4.2	Joint Placement	80
5.4.3	Accuracy of Joint Estimators	84
5.4.4	Consistency of Joint Estimators	84
5.4.5	Speed of Joint Estimators	89
5.5	Summary	89
6	Support Vector Machines	91
6.1	Basic Support Vector Machine	91
6.1.1	Linear Discriminant Function	93
6.1.2	Preliminary Equations and Constraints	94
6.1.3	Error Margin Formulation	95
6.1.4	Optimising for Largest Margin	98
6.1.5	Lagrange Formulation	99
6.1.6	Solving \mathbf{w} and b	103
6.2	Soft Error Margin	104
6.3	Nonlinear Classifier	107
6.3.1	Feature Space Transform	107
6.3.2	Kernel Functions	108
6.4	Multiclass Classifier	111
6.4.1	One-against-all	111
6.4.2	One-against-one	113
6.5	Summary	114

7	Pose Classification	115
7.1	Joint Feature Extraction	115
7.1.1	Position Features	115
7.1.2	Transformed Features	116
7.1.3	Joint Angles	122
7.2	Pose Classifier	124
7.2.1	Model Description	124
7.2.2	Model Training	125
7.3	Testing	125
7.3.1	Experimental Setup	126
7.3.2	Pose Classifier Accuracy	126
7.3.3	Confusion Properties	128
7.3.4	McNemar Tests	129
7.3.5	Pose Classifier Speed	132
7.4	Summary	132
8	Real World Testing	134
8.1	Experimental Setup	134
8.2	System Accuracy	135
8.3	Confusion Properties	136
8.4	McNemar Tests	141
8.5	System Classification Speed	141
8.6	Summary	143
9	Conclusion	144
9.1	Project Discussion	144
9.1.1	Region Classifier	144
9.1.2	Joint Estimator	145
9.1.3	Pose Classifier	145
9.1.4	Complete System	146
9.2	Similar Work	146
9.3	Future Work	147
9.3.1	Motion Recognition	147
9.3.2	Custom 3D Hand Modeller	147
9.3.3	Depth Sensors and Filters	148
9.3.4	Decision Forests	148
9.3.5	Dot Products as Joint Features	148
9.3.6	GPU Acceleration	148
	Bibliography	150

A	American Sign Language Digit Set	156
A.1	Table of ASL Digits	156
A.2	Tables of ASL Digit Images	157
B	Result Tables and Matrices	159
B.1	Region Classifier Results	159
B.2	Joint Estimator Results	165
B.3	Pose Classifier Results	173
B.4	Real World Results	178

List of Figures

1.4.1	Gesture Recognition (Brief)	5
1.4.2	Gesture Recognition (Detailed)	6
2.1.1	Illustration of the Kinect depth image noise and artefacts. It can be seen that the edges of objects are jagged, such as the edge of the hand in the two images. Also, there are discontinuities on the surface of objects caused by the “shadows” of objects closer to the sensor being projected onto the more distant objects. These shadow effects can be seen on the palm of the hand, where the little finger and thumb cause parts of the palm to become unrecognised by the sensor.	15
2.3.1	Overview of the complete pose recognition system.	21
2.3.2	The ASL numerical digits (from left to right): 1, 2, 3, 4 and 5.	23
2.3.3	The ASL alphabetical digits (from left to right): A, S, L, Y and W.	24
2.3.4	The ASL alphabetical digits O (left) and C (right). The top row shows the front view of the digits, while the bottom row shows the side view.	24
2.3.5	The “open” (top) and “closed” (bottom) variations of the ASL digits (from left to right): 2, 3, 4, 5 and W.	25
3.1.1	An example of a simple Decision Tree for identifying different animals.	29
3.2.1	General training procedure for a decision tree.	31
3.2.2	The modified decision tree.	34
3.2.3	Illustration of how the Decision Tree from Figure 3.1.1 splits the feature space (flattened to two dimensions).	37
3.3.1	An example problem showing how a combined classifier can prevent the effects of overfitting. The two outliers of the circle and cross classes are marked in bold. The green and red striped lines show the decision boundaries of the two classifiers trained using different subspaces of the data set (XY- and Y-subspaces respectively). The solid blue line shows the combined classifier found by averaging the previous classifiers.	39
4.1.1	Example of a labelled and a depth image.	45

4.1.2	The pose constraints placed on the system, where the z-axis is perpendicular to the palm and the x- and y-axes are co-planar to the palm.	48
4.2.1	Illustration of the offset vectors \mathbf{u} and \mathbf{v} from the pixel located at \mathbf{x} . The offset pixels used for depth comparison is marked as \mathbf{x}_u and \mathbf{x}_v	49
4.2.2	The scheduler used to manage threads.	51
4.3.1	Results of the Large and Smallest Region classifiers for both training sets A and B on testing set B. The first graph shows the per class results while the second graphs shows the same results that are weighted based on the number of pixels present for each class.	57
4.3.2	Results of the Large and Smallest Region classifiers for both training sets A and B on testing set A. The first graph shows the per class results while the second graphs shows the same results that are weighted based on the number of pixels present for each class.	59
4.3.3	Results of the Large and Smallest Region classifiers for both training sets A and B on testing set B. The first graph shows the per class results while the second graphs shows the same results that are weighted based on the number of pixels present for each class.	60
4.3.4	Speed test of the various classifiers trained using training set A.	61
4.3.5	The labels of the various joints corresponding to the 21 regions.	62
4.3.6	Confusion matrices of RDF consisting of 16 trees trained using the Large data set of Training Set A. The top confusion matrix shows the results of testing the classifier on Testing Set A, while the bottom matrix shows the results while testing with Test Set B.	64
4.3.7	Confusion matrices of RDF consisting of 16 trees trained using the Large data set of Training Set B. The top confusion matrix shows the results of testing the classifier on Testing Set A, while the bottom matrix shows the results while testing with Test Set B.	65
5.3.1	The green rectangle represents the smallest window needed to enclose all the foreground pixels in the given label image.	73
5.3.2	The effect of an adjacent joint a (dot) on the likelihoods of class c in the neighbourhood surrounding joint a	76
5.3.3	The effect of an adjacent joint a (dot) and a non-adjacent joint n on the likelihoods of class c in the neighbourhood surrounding the joints. The horizontal line in the sketch indicates the maximum of $M_c(\mathbf{x})$	77
5.3.4	The effect of two adjacent joints a_1 and a_2 (dots) on the likelihoods of class c in the neighbourhood surrounding the joints. The horizontal line in the sketch indicates the maximum of $M_c(\mathbf{x})$	77

5.4.1	Examples of depth images with noise applied. The top row shows the original depth images, while the bottom row shows the images with noise applied.	80
5.4.2	Graphs of the number of joints found missing using various region classifier and joint estimator combinations. The top graph shows the results when using region classifier RDF16_AT, while the bottom graph shows RDF16_AL. (Note difference in y-axis scale.)	81
5.4.3	Examples of poses where joints are placed on the background. (Misplaced Joints – Left: R2, M0, M1; Right: M0, I2)	82
5.4.4	Graphs of the number of joints misplaced using various region classifier and joint estimator combinations. The top graph shows the results when using region classifier RDF16_AT, while the bottom graph shows RDF16_AL. (Note difference in y-axis scale.)	83
5.4.5	Average distances between the estimated joint sets from the three joint estimators and the GS Centre of Gravity joint set. The top graph shows the results when using region classifier RDF16_AT to classify the depth images, while the bottom graph shows RDF16_AL.	85
5.4.6	Average distances between the joint sets estimated from the noise testing set and the Gold Standard joint set.	86
5.4.7	Standard deviation between the estimated joint sets from the three joint estimators and the GS Centre of Gravity joint set. The top graph shows the results when using region classifier RDF16_AT to classify the depth images, while the bottom graph shows RDF16_AL.	87
5.4.8	Standard deviation between the joint sets estimated from the noise testing set and the Gold Standard joint set.	88
5.4.9	The speed measurements for various combinations of region classifiers and joint estimators. The top graph shows the speed measurements for region classifiers trained using the Smallest subset of Training Set A, while the bottom graph shows the measurements for the Large subset.	90
6.1.1	Illustration of how the discriminant function classifies a 2D linearly separable two-class problem using a hyperplane. Figure S shows the original unlabelled data set. Figures A, B and C show how different hyperplanes will label the data set. Notice that the hyperplanes of B and C give the same classification, but C leaves more room for error.	94
6.1.2	Illustration of how the hyperplane of the discriminant function classifies a data set. The points enclosed in a square are the Support Vectors, which determine the size of the error margin as shown in blue.	96

6.1.3	A hyperplane and the corresponding \mathbf{w} vector. The points labelled \mathbf{x}_1 and \mathbf{x}_2 lie on the plane and are connected by the vector \mathbf{l}	96
6.1.4	Illustration of the vectors that are used to calculate the Euclidean distance between any point and the hyperplane.	97
6.2.1	Illustration of the soft margin and slack variables. The support vectors now include data points inside the margin.	105
6.3.1	Example of how a transform from the \mathcal{X} -space to the \mathcal{Z} -space could make a nonlinear problem linear.	107
6.4.1	Example of a One-against-all multiclass classifier. The top row shows 3 potential classifiers trained from the three 2-class problems, including the direction of the \mathbf{w} vector. The yellow data points represent the negative examples for each of the individual classifiers. The bottom row shows the combined classifier along with the indeterminate regions and the final classification regions.	112
6.4.2	Example of a One-against-one multiclass classifier. The top row shows 3 potential classifiers trained from the three 2-class problems. The grey data points indicate the data which are not considered during the training of an individual classifier. The bottom image shows the decision boundaries of the combined classifier. The yellow area indicates an indeterminate area, where all classes receive an equal number of votes when classifying a new data point inside the region.	113
7.1.1	A simple translation transformation.	117
7.1.2	The rotation transformations.	118
7.1.3	The scaling transformation. P, R, M, I and T indicate the second joints of the five fingers and O indicates the palm joint.	119
7.1.4	The angle $\phi_{\mathbf{A},\mathbf{B}}$ between three joints \mathbf{i} , \mathbf{j} and \mathbf{k}	122
7.1.5	A view of the joints of a bent index finger, the palm and wrist from the side, along with a few example angles used in the feature set.	123
7.3.1	The results of testing the various pose classifiers on Testing Set A. The two graphs show the results when using Region Classifiers RDF16_AT (top) and RDF16_AL (bottom).	127
7.3.2	The results of testing the various pose classifiers on Testing Set B. The two graphs show the results when using Region Classifiers RDF16_AT (top) and RDF16_AL (bottom).	128

7.3.3	Confusion colour graphs indicating which poses the system confuses with which. The two graphs show the results when using the best performing pose classifier (RDF16_BL, Reservation, Transform3D) on Testing Set A (top) and Testing Set B (bottom). Table B.3.6 and Table B.3.7 show the corresponding confusion matrices.	130
7.3.4	McNemar tests to evaluate the comparisons of Figure 7.3.2 (bottom). . . .	131
7.3.5	The results of speed tests performed on the system using the synthetic data from Testing Set B.	132
8.2.1	Real world recognition rate of the pose classifiers built using the RDF16_BL Region Classifier with various joint estimators and feature sets. The results include the recognition rate from pose classifiers trained using the synthetic data set and the real world data set.	136
8.3.1	Confusion colour graphs indicating which poses the system confuses with which. The two graphs shows the results when using the best performing pose classifier (RDF16_BL, Reservation, Transform3D). The top graph shows the results of the pose classifier using the SVM models trained with the synthetic data, while the bottom graph shows the results of the pose classifier trained using the real world data.	138
8.5.1	The classification speed of the complete Pose Recognition System.	141
8.5.2	McNemar tests to evaluate the comparisons of Figure 8.2.1 (bottom). . . .	142

List of Tables

3.2.1	The average length of a rule for the original and the modified decision trees shown in Figure 3.1.1 and Figure 3.2.2	34
3.2.2	The number of data points belonging to each class.	37
4.3.2	Constraints placed on the generated training and testing sets.	54
4.3.3	The various region classifiers trained and tested.	55
4.3.4	Number of images per RDT in the subsets of Set A and Set B.	56
5.4.1	Table showing the total joints lost or misplaced by the joint estimators. . .	82
5.4.2	Table summarising the accuracy and consistency results of the joint estimators. The best results for each test are marked in bold.	88
8.3.1	Confusion matrix showing the recognition rates of the different poses of the real world testing set using Region Classifier RDF16_BL, the Reservation Joint Estimator and the Transform 3D joint features, with SVM models trained using synthetic data. Empty entries indicate values smaller than 0.01.	139
8.3.2	Confusion matrix showing the recognition rates of the different poses of the real world testing set using Region Classifier RDF16_BL, the Reservation Joint Estimator and the Transform 3D joint features, with SVM models trained using real world data. Empty entries indicate values smaller than 0.01.	140
A.1.1	The list of 17 poses used to train and test our system, which includes 12 ASL digits of which 5 digits have two variations.	156
A.2.1	The gesture set used to train the system (Numbers).	157
A.2.2	The gesture set used to train the system (Letters).	158
B.1.1	Table summarising the recognition rate of various Region Classifiers when tested against Testing Set A.	159

B.1.2	Table summarising the recognition rate of various Region Classifiers when tested against Testing Set A, weighted according to region representation in testing set.	159
B.1.3	Table summarising the recognition rate of various Region Classifiers when tested against Testing Set B.	160
B.1.4	Table summarising the recognition rate of various Region Classifiers when tested against Testing Set B, weighted according to region representation in testing set.	160
B.1.5	Table summarising the classification speed for various Region Classifiers.	160
B.1.6	Confusion matrix showing the recognition rates of the different hand regions when testing RDF16_AL on Testing Set A. Empty entries indicate values smaller than 0.01.	161
B.1.7	Confusion matrix showing the recognition rates of the different hand regions when testing RDF16_AL on Testing Set B. Empty entries indicate values smaller than 0.01.	162
B.1.8	Confusion matrix showing the recognition rates of the different hand regions when testing RDF16_BL on Testing Set A. Empty entries indicate values smaller than 0.01.	163
B.1.9	Confusion matrix showing the recognition rates of the different hand regions when testing RDF16_BL on Testing Set B. Empty entries indicate values smaller than 0.01.	164
B.2.1	Table summarising the speed of the various Joint Estimators when using a Region Classifiers trained using the Smallest training subset.	165
B.2.2	Table summarising the speed of the various Joint Estimators when using a Region Classifiers trained using the Large training subset.	165
B.2.3	Table summarising the percentage of joints not found for each joint estimator.	166
B.2.4	Table summarising the percentage of joints not placed on the surface of the hand for each joint estimator.	167
B.2.5	Table summarising the average distance from the Gold Standard each joint estimator placed joints, when testing against Testing Set A.	168
B.2.6	Table summarising the standard deviation from the Gold Standard each joint estimator placed joints, when testing against Testing Set A.	169
B.2.7	Table summarising the average distance from the Gold Standard each joint estimator placed joints, when testing against Testing Set A.	170
B.2.8	Table summarising the standard deviation from the Gold Standard each joint estimator placed joints, when testing against Testing Set A.	171

B.2.9	Table summarising the average distance from the Gold Standard each joint estimator placed joints, when testing against Testing Set A with the shuffled algorithm applied to add noise similar to that of the Microsoft Kinect.	172
B.2.10	Table summarising the standard deviation from the Gold Standard each joint estimator placed joints, when testing against Testing Set A with the shuffled algorithm applied to add noise similar to that of the Microsoft Kinect.	172
B.3.1	Table summarising the recognition rate of various pose classifiers when tested on Testing Set A, using 16-tree region classifiers trained using the Smallest subset from both Training Set A and B.	173
B.3.2	Table summarising the recognition rate of various pose classifiers when tested on Testing Set A, using 16-tree region classifiers trained using the Large subset from both Training Set A and B.	173
B.3.3	Table summarising the recognition rate of various pose classifiers when tested on Testing Set B, using 16-tree region classifiers trained using the Smallest subset from both Training Set A and B.	174
B.3.4	Table summarising the recognition rate of various pose classifiers when tested on Testing Set B, using 16-tree region classifiers trained using the Large subset from both Training Set A and B.	174
B.3.5	Table summarising the classification speed of various pose classifiers when tested on Testing Set A, using 8- and 16-tree region classifiers trained using the Large subset of Training Set B.	175
B.3.6	Confusion matrix showing the recognition rates of the different poses of Testing Set A using Region Classifier RDF16_BL, the Reservation Joint Estimator and the Transform 3D joint features. Empty entries indicate values smaller than 0.01.	176
B.3.7	Confusion matrix showing the recognition rates of the different poses of Testing Set B using Region Classifier RDF16_BL, the Reservation Joint Estimator and the Transform 3D joint features. Empty entries indicate values smaller than 0.01.	177
B.4.1	Table summarising the recognition rate of various pose classifiers when tested on real world data, using RDF16_BL as the region classifier and pose classifiers trained using synthetic and real world data.	178
B.4.2	Table summarising the classification speed of various pose classifiers when tested on real world testing data, using 8- and 16-tree region classifiers trained using the Large subset of Training Set B and pose classifiers trained using the real world training data.	178

Nomenclature

Definitions

Class	Label used for a group of data points with the same identity or meaning.
Feature	A term derived from the field of Computer Vision. It describes an important piece of information used to model a specific problem.
Detector	A mathematical element used to find important or unique features within a data set. The exact element used depends on the application.
Descriptor	A mathematical element used to describe a feature in order to compare features from the same feature set. The exact element used depends on the application.
Pose	A description of the position and orientation of an object.
Body Pose	A description of the orientation of the body and limbs as a whole.
Hand Pose	A description of the orientation and shape/sign of the hand, i.e. the palms and fingers.
Gesture Motion	A description of the movement of the hand, as caused by the wrists and arms. In this document, the gesture motion describes the complete movement over a sequence of frames, which can include simple movements (straight lines or curves) or more complex movements (depiction of characters or numbers).

Hand Gesture	In the literature, hand gestures are used to describe either the hand pose or the motion made by the hand. In this document, the hand gesture describes the combination of the hand pose and motion in order to convey information.
Ground Truth	The correct or ideal output of a certain process or function. Used to determine the accuracy of different algorithms that perform the same function using different methods, by comparing the output from the algorithm with that of the ground truth.
Confusion Matrix	A summation table plotting the ground truth against the actual output of an algorithm for various inputs. Used to compare the accuracy of the outputs given various inputs for different algorithms.
Region Classifier	The software component of this project which is responsible for classifying the pixels of a depth image to their corresponding hand regions.
Hand Region	A labelled region of the hand associated with one joint and represented using a group of pixels.
Joint	Normally refers to the connection point between two bones, but used in this document to reference either the centre of a finger bone or the centre of gravity of either the palm or wrist.
Joint Estimator	The software component of this project which is responsible for estimating the hand 3D coordinates of the hand joints from a labelled hand image.
Joint Features	A set of features extracted from a set of 3D joint coordinates and used for classifying the hand pose.
Pose Classifier	The software component of this project which is responsible for classifying the hand pose from a set of joint features.
Split Criterion	The test used to split a data set into separate subsets.

x-axis, y-axis	The two image axes, where the x-axis is associated with the width of the image and the y-axis is associated with the height of the image.
z-axis	The axis associated with the depth component of a scene.

Abbreviations

ASL	American Sign Language
COG	Centre of Gravity
CRF	Conditional Random Field
CPU	Central Processing Unit
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
KKT	Karush-Kuhn-Tucker
LIDAR	Light Detect and Ranging
RC	Region Classifier
RDF	Random Decision Forest
RDT	Random Decision Tree
ROI	Region of Interest
SDK	Standard Development Kit
SEP	Start and End Point
SVM	Support Vector Machine

Symbols – Common

x	An n-dimensional feature vector.
\mathbf{v}^T	The transpose of the vector v .
d	Number of dimensions.
\in	Element of ... / In set ...
\forall	For all ...
\cdot	Absolute value

$\| \cdot \|$ Vector norm / magnitude

$\frac{\delta}{\delta \cdot}$ Partial derivative

$\lceil \cdot \rceil$ Ceiling

$\lfloor \cdot \rfloor$ Floor

Symbols – Chapter 3

x_i The scalar value of the feature vector \mathbf{x} for the i^{th} dimension.

τ A scalar threshold value.

\mathbf{a} An n-dimensional weight vector.

a_i The scalar value of the weight vector \mathbf{a} for the i^{th} dimension.

S A set of data points.

$|S_A|$ The number of data points in set A.

$H(S_A)$ Shannon Entropy of set A.

ϕ Split criterion, occasionally shortened as split.

$S_A(\phi)$ Subset A formed using split criterion ϕ .

$f_{split}(S_A, \phi)$ Decrease in entropy of set A using split criterion ϕ .

\mathbf{p} An n-dimensional probability vector.

p_i The scalar value of the probability vector \mathbf{p} for the i^{th} dimension.

c Data class/label.

Symbols – Chapter 4

\mathbf{P}	The 2D vector containing the image coordinates of pixel P .
\mathbf{u}, \mathbf{v}	2D offset vectors.
$I(\mathbf{x})$	Depth value of pixel located at 2D coordinates \mathbf{x} of depth image I .
$F_{\mathbf{u},\mathbf{v}}(I, \mathbf{x})$	Scalar feature value at pixel \mathbf{x} of image I , given offset vectors \mathbf{u} and \mathbf{v} .

Symbols – Chapter 6

\mathbf{w}	The SVM classifier weight vector. Determines the orientation of the hyperplane.
b	The bias scalar value of the SVM classifier.
\mathbf{s}_i	Support vector i .
N_s	Number of support vectors.
$\mathbf{l} \text{ [L]}$	Line vector joining two points on the hyperplane.
$D_{\mathbf{w},b}(\mathbf{x})$	Euclidean distance between point \mathbf{x} and the hyperplane described by \mathbf{w} and b .
$\hat{\mathbf{w}}$	The unit vector of the weight vector \mathbf{w} .
y	Class label.
\mathbf{a}	The argument list of the Lagrange function.
λ	The vector of Lagrange Multipliers.
$\mathcal{L}(\mathbf{a})$	The Lagrange function with argument list \mathbf{a} and Lagrange Multipliers λ .

\mathbf{Q}	Matrix containing features and labels for quadratic programming.
$\mathbf{0}$	Vector/Matrix containing all zeros.
$\mathbf{1}$ [One]	Vector containing all ones.
\mathbf{x}_s	A particular support vector.
\mathbf{x}_h	The intersection point between the hyperplane and the weight vector \mathbf{w} .
ξ	Slack variable vector.
C	Penalty parameter of soft margin SVMs.
μ	Slack variable Lagrange Multipliers vector.
$\mathcal{X}, \mathcal{Z}, \mathcal{R}$	Features spaces.
\mapsto	Maps to ...
Φ	Transformation from \mathcal{X} -space to \mathcal{Z} -space.
\mathbf{z}	Feature vector in the \mathcal{Z} -space.
$K(\mathbf{x}', \mathbf{x})$	Kernel function.
M	Number of classes.

Symbols – Chapter 7

\mathbf{f}	Feature vector.
\mathbf{T}	Transpose matrix.
$\mathbf{R}_x, \mathbf{R}_y, \mathbf{R}_z$	Rotation matrices around the x-, y- and z-axes respectively.
\mathbf{S}	Scaling matrix.

\mathbf{A}	Combined transformation matrix.
\mathbf{X}	Homogeneous feature matrix.
$\phi_{\mathbf{A},\mathbf{B}}$	Angle between vectors \mathbf{A} and \mathbf{B} .
γ	libSVM Gaussian kernel function parameter.

Chapter 1

Introduction

Natural interaction of computer systems is an open research problem which has received a large amount of attention with the introduction of cost-effective depth sensors such as the Microsoft Kinect. This thesis focuses on the natural interaction of a computer system using hand gesture recognition. Specifically, we focus on the use of depth images to create a real-time system which circumvents many of the common computer vision problems and is accurate and robust.

This document presents the research performed during this project, providing the relevant theory needed to create a gesture recognition system. The document details the system design and show how the final system was evaluated. The test results shows that the final system has an accuracy of 95.61% when tested on synthetic data, which drops to 81.35% when tested on real world data.

The start of the chapter discusses the background and applications of a complete gesture recognition system. Later sections focus on the hand tracking and pose recognition components of a gesture recognition system, which is the main focus of the project.

1.1 Gesture Recognition System Applications

There are many possible applications for a hand gesture recognition system. The primary goal of such a system is to create an intuitive input interface, where the user can communicate commands to a computer system without the need for any apparel or electronic devices.

An important application is the translation of sign languages. A translation device can be used to facilitate communication between different sign languages and parties

who have no experience in any sign language. It can also be used as a teaching aid to help learn different sign languages.

A gesture recognition system can also be used to remotely control electronic household appliances without the need for a physical remote control. The user would be able to change the volume or channel on their television set, change the brightness of the lights or control the output temperature of the air conditioning unit, simply by using hand gestures. Freeman et al. [1] demonstrated a system to control a television set using very simple hand gestures. Microsoft used gestures together with the Kinect to control video games without the need of remote controls [2].

Another important application is touch-free interfaces inside clean-rooms, such as operating theatres and microelectronic laboratories. A gesture interface could remove the need to clean touch interfaces for sanitary reasons, saving money and time on cleaning these interfaces.

1.2 Existing Systems

A few examples of gesture recognition systems already exist. This section discusses two of these systems, one developed by GestureTek and another by Microsoft. These two systems are described below.

1.2.1 GestureTek

GestureTek (www.gesturetek.com) is a company which specialises in developing products which use various forms of gestures to communicate with computer systems. The GestTrack3D Standard Development Kit (SDK) is of particular interest. The SDK provides functions to track the hands of a user and recognise various motions made by the hands ("Swipe", "Poke", "Steer", etc.). Products have been developed to create interactive billboards located at shop windows which clients can control using hand gestures and to emulate mouse control of desktops. The hand pose recognition capabilities of these systems are limited to recognising an open palm and pointing gesture.

1.2.2 Microsoft Kinect

The Microsoft Kinect is a colour and depth camera in one device developed for the Xbox gaming console. Previously, depth images had to be calculated using a stereo camera setup [3, 4] or using Light Detection and Ranging (LIDAR) devices [5, 6]. A stereo camera setup needs to be carefully calibrated by researchers (all calibration of

the Kinect relating to depth images are done at the factory level) and requires large amounts of computational resources to extract the depth image in real-time, while LIDAR requires expensive equipment which is not ideal for general use. The Kinect provides an inexpensive solution of retrieving the depth image in real-time using an infrared projector and camera setup.

The Kinect was originally designed as part of a gesture recognition system to enable control of video games on the Xbox without physical controllers. This is accomplished by estimating the pose of the user's body [2], but only the general location of the hands are retrieved. With the introduction of open source drivers and libraries (www.openni.org) and the official Microsoft Kinect SDK (www.kinectforwindows.org), other applications have been developed [7, 8, 9, 10]. The real-time hand pose recognition of [7] which is based on [2] is of particular interest to this project, since they gave implementation details of their working pose recognition system.

The Kinect is used in this project to retrieve the real world images which are used to test our system, because it is cost-effective and can easily be acquired by consumers. The following section will discuss in more detail how the Kinect was used for body and hand pose recognition in [2] and [7]. The subsequent section discusses the work that has been done on camera calibration of the Kinect.

Body Pose Estimation

Microsoft developed an algorithm for the Kinect to extract a skeleton model of a person from a depth image [2]. This skeleton model is used in games written specifically for the Kinect to track the motion of the various joints of the user. This enables the program to estimate the pose of the user's body, which allows for a more immersive control scheme for the Xbox.

The program attempts to identify the pose of the complete body by determining the pose of the smaller sections. The technique first separates the user from the background using the depth images from the Kinect. A set of three different feature detectors are applied to the resulting image. The results of each detector are stored in a separate Random Decision Tree (RDT) [11]. The three RDTs are then combined into a Random Decision Forest (RDF) [12, 13].

Finally, the RDF is used to determine what section each pixel in the image represents. The location of the skeletal joints can be found using these sections and adding a learned depth offset to place the joints inside the body.

Hand Pose Recognition

The system of [7] applies the above algorithm on the user's hands for pose recognition. Their approach gave more attention to occlusions because of the frequency that self-occlusions occur when communicating with the hand. The paper presented a simple solution: the location of the occluded joints were simply moved to the nearest connected joint. This proved to provide a strong descriptor for recognition purposes.

The technique was evaluated using a data set consisting of 10 American Sign Language (ASL) signs performed by 10 different people. The system achieved a recognition rate of 99.9% at a frame rate of 30Hz, which is adequate for this project.

Camera Calibration

The images retrieved from the colour and depth cameras of the Kinect are not exact copies of the scene. This is the case with any camera, as imperfections in the manufacturing process can cause various distortions on the final image. The colour and depth images also do not correspond pixel for pixel, because each camera is subjected to different distortions and located at different positions. The distortion and misalignment of the images can be removed using the process of camera calibration.

The calibration process determines the internal parameters of the camera, which are used to remove distortions from the images captured by the camera. Calibration of colour cameras has been thoroughly researched and a few calibration techniques have been developed [14, 15, 16]. However, these techniques cannot be used to calibrate the depth camera of the Kinect. Zhang et al. [17] and Herrera et al. [18] both developed techniques to calibrate a colour and depth camera stereo pair which can be used to calibrate the Kinect.

1.3 Research Problem

The problem addressed in this research is the implementation of a real-time hand tracking and pose recognition software system using single view depth images, which forms part of a larger gesture recognition system.

1.4 Gesture Recognition Problem

Hand gesture recognition (referred to as gesture recognition henceforth) refers to the problem of interpreting the movement and pose of a user's hands from a sequence of

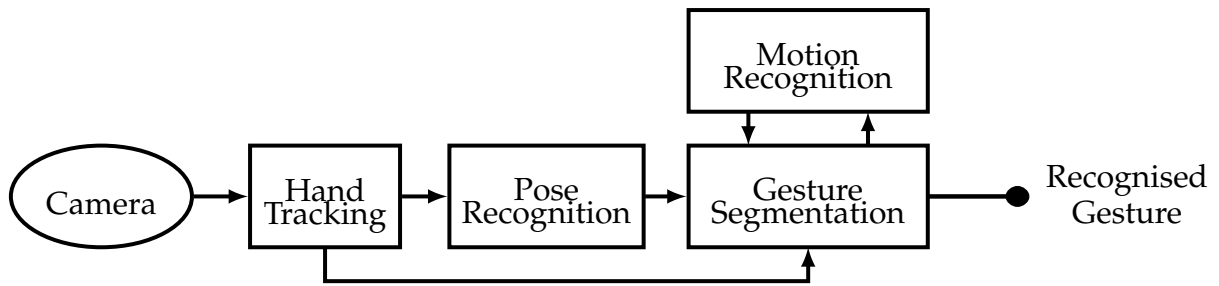


Figure 1.4.1: Gesture Recognition (Brief)

images using a computer system. Such a system enables the user to control a computer system using hand gestures if a camera is connected to the system to provide a video feed. The problem is actively being researched in the field of Computer Vision [19, 20, 21].

Gesture recognition can be divided into four phases, which are shortly described below and illustrated in Figures **Figure 1.4.1** (brief) and **Figure 1.4.2** (detailed). The four phases are Hand Tracking, Pose Recognition, Motion Recognition and Gesture Segmentation.

Hand Tracking: The 2D coordinates of the centre of the user's hands are found within each frame in a video sequence during the hand tracking phase. These coordinates are represented by the Centre of Gravity (COG) of the hand and are used to determine a region of interest (ROI) within each frame. The 2D coordinates (or 3D coordinates if depth data is available) of the COG are used for motion recognition, while the ROI is used for pose recognition.

Pose Recognition: Pose recognition is the process of converting a 2D image of a hand into a label describing the pose (e.g. "open", "fist", "pointing", etc.). The process can either classify the pose directly from the image (pixel-based) or from extracted features (feature-based). The ROI representing the hand within each frame is used to recognise the pose. The final result is a label representing the pose of the hand for each frame.

Motion Recognition: The complete motion of a hand for a sequence of frames is classified during this phase. It is assumed that a given sequence contains only one significant motion. A set of sequential 2D or 3D coordinates of the COG is used to determine if a meaningful motion was made by the user. The final result is a label representing the motion for the sequence (e.g. "left-to-right", "circle", "number 3", etc.).

Gesture Segmentation: Gesture segmentation describes the process of separating the meaningful gestures in a sequence of frames from the idle gestures. The start and end points (SEP) of potentially significant gestures are extracted from a sequence of frames. The set of 3D coordinates of the COG between these points is used to recognise the motion of the subsequence. The final gesture for the subsequence is then determined using the hand pose labels at the SEP and the motion label. The output consists of the label for the gesture and the SEP of the gesture sequence.

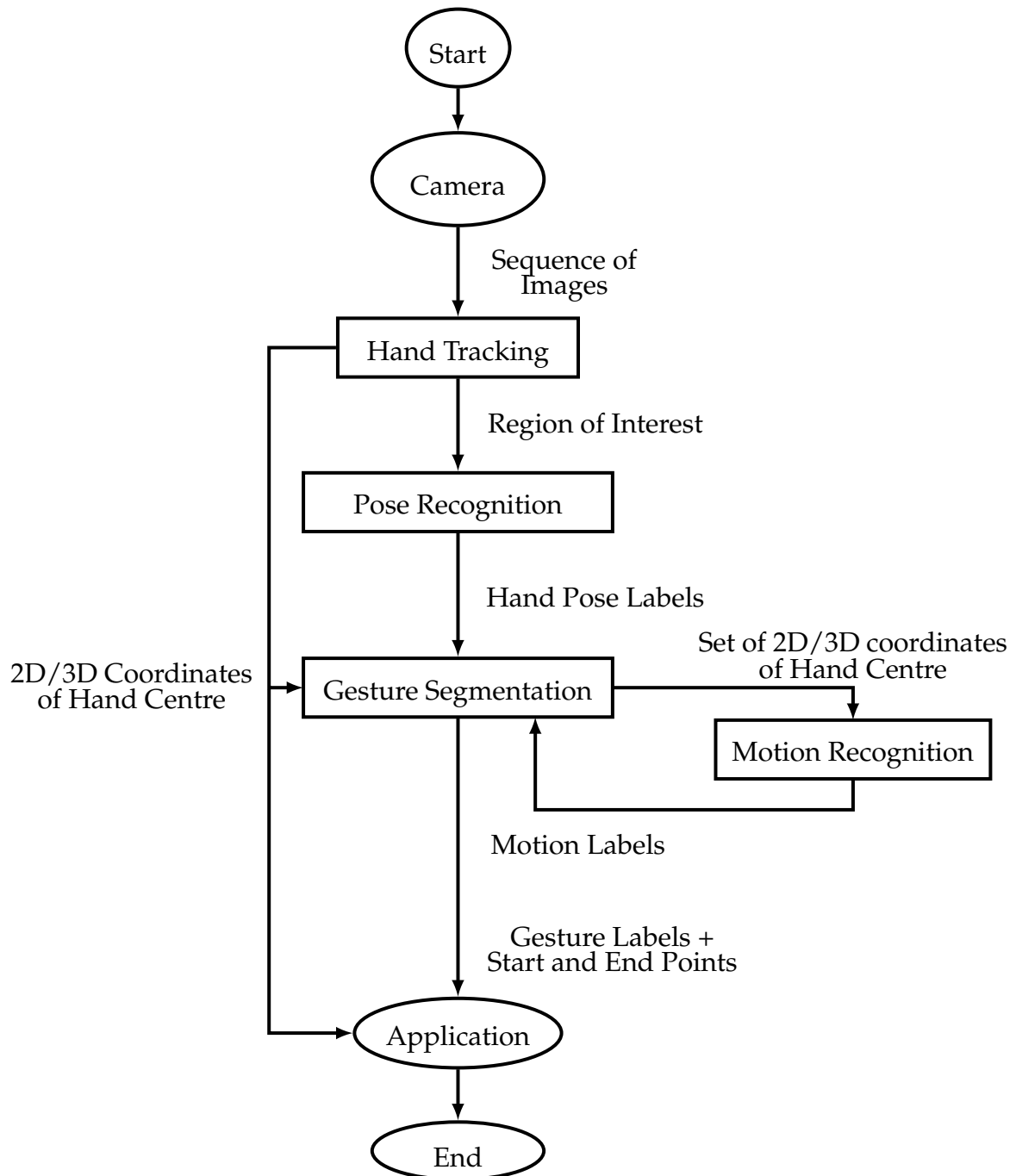


Figure 1.4.2: Gesture Recognition (Detailed)

These four components are all needed for a complete gesture recognition system. This project however only implements the hand tracking and pose recognition components. The following two sections discuss these two components in more detail.

1.4.1 Hand Tracking

The first objective of a gesture recognition system is to find the hands of the user. The difficulty of this task depends on the current environment. It can be difficult to track the user's hands against a cluttered background. This is due to the difficulty of creating a mathematical model to uniquely identify the human hands. In order to overcome these problems, a few techniques of hand tracking have been proposed. A selection is presented below.

Colour-based and Marker Detection

As the name suggests, these techniques make use of colour information to find the user's hands. In [22], a brightly coloured marker is placed on the hand of the user. The colour of the marker is chosen in such a way that it can be easily distinguished from the rest of the scene. The location of the user's hands in an image can then be found by searching for the marker, which is identified by its colour. Gloves are sometimes used instead of markers [23], which eases the process of retrieving the hand surface.

Another similar technique extracts the skin-coloured regions from the scene for tracking purposes [24, 25, 26]. The image is scanned for pixels that lie between a predetermined colour range. The potential skin pixels are then grouped together based on a predetermined criteria to form regions. These regions are then used to find the hands and face of the user.

Colour-based techniques are simple to implement and are fast to compute, but at the cost of accuracy and robustness. The accuracy of these techniques can vary between different backgrounds and lighting conditions. This is caused by the detector finding a colour similar to the marker or skin of the user within the background. This will cause the system to mistakenly detect hands in the background.

Example-based

Image features have successfully been used for object recognition [27], [28]. Normally image features are extracted from an image using a gradient function, such as Sobel Filters or Haar-wavelets. The hand's ability to deform into various shapes and poses can cause some difficulty in successfully finding features. The number and type of

features detected on a hand can change considerably between images, as the hand is rotated and changes shape. The background of the scene can also produce false positives similar to the colour-based techniques.

In [29], Haar-like features are used to create a hand detector. The detector performs well to changes in translation and scale. However, rotation does provide some difficulty to the detector as the local image area of a feature can change considerably with rotations.

Body Pose Estimation

Another approach to the problem of hand tracking is to determine the full pose of the user's body. The pose of the body is usually described using a skeleton model, which contains the 3D position of various skeletal joints, including the wrists of the user. The 3D coordinates of the wrists can be used to estimate the location of the user's hands within a corresponding image.

Earlier vision-based techniques used colour images to estimate the pose of the body. These techniques include Shape-from-Silhouette [30] and 3D Modelling-based [31]. The biggest obstacle these techniques have to overcome is detecting the presence and location of a body. This can be difficult to accomplish when there are many other objects present in the scene.

Depth images have been considered more recently for body pose estimation. The Microsoft Kinect extracts and uses the depth images of a scene to estimate the pose of the user [2]. This information is used to find the hands of the user.

1.4.2 Pose Recognition

Once the hands of the user have been located, the next step is to determine and label the pose of the hands. Pose recognition is complicated by the fact that the hand has many degrees of freedom. This makes modelling the hand extremely difficult, as the hand can be rotated and deformed in many different ways. Self-occlusion is another problem, where the fingers or palm of the hand can block the view of a part of the hand. This can cause ambiguities in the exact pose of the hand. A few of the current techniques of pose estimation are discussed below.

Example-based

As discussed in Section 1.4.1, local features can be used to track the hand. The principles for recognising the pose are similar, with the only real difference being the end goal. When tracking the hand, the system only needs to recognise where the location of the hand is. With pose estimation, the system has to differentiate between the different poses. Features are extracted from the ROI and used to determine the correct pose label for the image. Since mostly the same principles are applied for pose recognition, we will refer back to Section 1.4.1.

3D Modelling

Another approach to determine the pose of the hand is to first build a 3D model of the hand [32, 33, 7]. The model contains the 3D position and orientation of each finger joint and the palm with respect to the wrist. This model can then be used to correctly label the pose of the hand. This approach is more intuitive than an example-based approach, but can be more difficult to implement.

In [32], a 3D model of a hand is extracted from the scene using a Point Distribution Model built from training images. Their system was able to calculate a 3D model of the hand at 18Hz, without the need for any markers. The system needed to be initialised by the user, by placing their hand over the centre of the video frame. The system was able to easily track translations and performed well under certain deformations of the hand. Rotations and changes in scale proved to be more difficult. The system was also not designed to handle occlusions, which are especially prevalent as self-occlusions of the hand.

More recently, [34] and [7] used the Kinect to extract a 3D model of the hand. The system developed by [7] extracted a 3D skeleton model of the hand in real-time, which also had a high tolerance for self-occlusions. The system and techniques used by [7] are described in Section 1.2.2.

1.5 Objectives

The primary objective of this project is to implement and test a real-time Pose Recognition system using depth images and the technique proposed by [7] with the introduction of new and improved joint features. Most of the existing gesture recognition systems only use hand motions for gesture recognition, as described in Section **Section 1.2**. However, a system which implements some form of pose recognition allows for a wider range of recognisable hand gestures, increasing the potential input

vocabulary of a gesture recognition system.

The secondary objective of this project is the development of a Kinect interface for extracting data from the device and the implementation of a real-time Hand Tracking component. The Kinect interface was implemented using OpenNI, an external library for communicating with natural interaction devices such as depth sensors. The hand tracking component was also implemented using external libraries in order to save time and allow us to focus on the primary objective. Two different external libraries were tested, namely OpenNI and Skeltrack, in order to determine the advantages and disadvantages of using each library in a gesture recognition system.

Only the interface to the Kinect, hand tracker and pose recogniser were implemented during the course of the project because of time constraints. The development of the motion recognition and gesture segmentation components is left for future projects.

The objectives are summarised as follows:

- Development of a software module for interfacing with the Kinect and retrieving the colour and depth images from the cameras using external libraries.
- Development of a software module for tracking the hands of a user using external libraries.
- Software implementation of a real-time Hand Pose Recognition system using depth information by implementing the technique proposed by [7].

1.6 Contributions

The following contributions are made by this project:

- We confirm the results given by Ho [12] for their combined classifiers, specifically using a collection of random decision trees in a forest to create a more accurate classifier.
- We confirm the results given by Keskin [2] for their proposed hand pose recognition system.
- We show how classifiers can be trained using a combination of synthetic and real world data and used to accurately classify real world data, specifically how computer generated and labelled hand depth images can be used to train a hand region classifier which can be used to train pose classifiers from real world data.

- We propose and test several different features extracted from a set of hand joints and used for pose classification, including Angle and Transform features. We find that our proposed Transform features perform better than using the joint coordinates directly for classification. We further find no significant difference between 2D and 3D features, and therefore propose using the 2D features for their speed.
- We test the difference in performance between three different joint estimators: Centre of Gravity, Mean-Shift and our own proposed modification of the Mean-Shift algorithm. We find that the Centre of Gravity estimators perform better than expected for a pose recognition system, though our Reservation estimator is more accuracy. McNemar tests do however show that further testing is required.

1.7 Thesis Overview

The thesis is divided into eight chapters. This chapter gives an introduction to the thesis and the research problem. The succeeding chapters describe the overview of the system and the theoretical basis and implementation of the various system components.

Chapter 2 gives an overview of the proposed Pose Recognition system. The chapter describes the three components found in our system, namely the Depth Sensor Interface, the Hand Tracker and the Pose Recogniser. The various constraints placed on the system is described in **Chapter 2** along with the motivation for each constraint. The chapter also indicates which external libraries are used to implement part of the system.

Chapter 3 and **Chapter 4** describe the theoretical basis and implementation of the Region Classifier used to classify the individual pixels of a depth image. **Chapter 3** gives the theoretical basis of Decision Tree classifiers and describes how the classifiers can be extended to improve the general performance by using a randomisation training algorithm and combining individual classifiers into a forest. The first half of **Chapter 4** describes the implementation the Region Classifier using the Decision Trees from **Chapter 3**. The second half provides the results of various tests performed to evaluate the implemented classifiers and discusses these results.

Chapter 5 describes the theory associated with the Mean-Shift algorithm and the implementation of a joint estimator using the algorithm. The chapter also describes our variation on the Mean-Shift algorithm, which uses heuristics specific to the hand

joint estimation problem, which attempts to improve the accuracy of the joint estimator. The chapter concludes with the results and discussion of the tests performed on the joint estimators for evaluation purposes.

Chapter 6 and **Chapter 7** discuss the theoretical basis and implementation of the Pose Classifier component of our system. **Chapter 6** discusses Support Vector Machines (SVM) and the extensions made to the basic SVM classifier to accommodate the multiclass, non-linear nature of the pose recognition problem. **Chapter 7** discusses the implementation of the Pose Classifier using SVMs trained using the external library called libSVM. The chapter also discusses the joint features we investigated for the project. **Chapter 7** ends with a description of the tests performed to evaluate the Pose Classifier and discussions of the results.

Chapter 8 shows the results of testing the system on real world data retrieved from the Microsoft Kinect depth sensor. The chapter discusses the environment in which the tests were performed and the final results of the testing. Both the accuracy and the speed of the final system are tested in this chapter.

Chapter 9 provides a brief summary of the results of the previous chapters. The chapter also discusses various possible improvements to the system and other future work.

Chapter 2

System Overview

This chapter gives an overview of the complete hand tracking and pose recognition system. It describes how our system retrieves data from a depth sensor, tracks a user's hands and finally classifies the pose of the hands. It will further describe the various constraints placed on the system and the assumptions, while also providing the details of the expected inputs and outputs of each component of the system.

2.1 Depth Sensor Interface

This section describes the Depth Sensor Interface created for this project. The interface is responsible for retrieving colour and depth data from a connected depth sensor for further processing. An external library was used to achieve this, namely the OpenNI [35] library. The Microsoft Xbox Kinect was used as the depth sensor for this project, though many of the data processing techniques mentioned is also applicable to other depth sensors.

2.1.1 OpenNI

OpenNI is an open source library developed to help promote development and research into the use of "Natural Interaction" devices. Natural Interaction refers to a way of interacting with computer systems which is more intuitive to humans than the tradition input devices, such as using speech or body language. Natural Interaction devices include Microphone arrays, such as found on the Microsoft Kinect for speech recognition, and depth sensors, which simplify the process of retrieving data from an observed scene.

The library was developed as an SDK to be used with any compatible devices, which at the time of writing include Microsoft's Xbox Kinect and Kinect for Windows, PrimeSense's Sensor and the ASUS's Xtion [35]. The SDK was chosen because it is

actively being developed. Version 1.5.4 was used during development of our system. Furthermore, it is open source and can be used across a wide range of platforms.

The version of OpenNI used during development did not include working drivers for the Xbox Kinect which we used for testing. This was corrected by using the SensorKinect [36] drivers available on GitHub, which is a fork of the PrimeSense Sensor module. These drivers needed to be used with an unstable version of OpenNI (v1.5.4), but should not be necessary for the newer versions.

2.1.2 Data Processing

A simple module was written to retrieve data from the Kinect. This module retrieves both the depth and colour data from the Kinect sensor. The colour data is stored in a 640x480 RGB image, while the depth data is stored in a 640x480 floating point image, though the actual resolution of the depth image is 320x240. The measurement units of the depth image are in mm and the values range from 700 to approximately 6000, which is the range of the depth sensor. A special value of zero is assigned to a pixel if the depth is unknown, either because it is out of the sensor's depth range or the sensor was unable to estimate the depth.

The depth data retrieved needs to be further processed to conform with the input constraints of the Pose Classifier and to remove noise and other artefacts from the image. The depth image is sent through two stages of processing, namely image correction followed by noise and artefact removal.

Depth Image Correction

The depth and colour images retrieved from the Kinect do not correspond pixel-for-pixel, even when the depth image is scaled up to the same resolution as that of the colour image. OpenNI provides a function to transform the depth image pixel space to the colour image space and vice versa. This transformation is not needed by the pose classifier, since the classifier only uses depth data and is designed to be invariant to the shape of the hand. We did however use it for debugging and demonstration purposes.

During this stage we also convert the units of the depth values from mm to metres. This is done to comply with the constraint the pose classifier places on the input depth image's format.

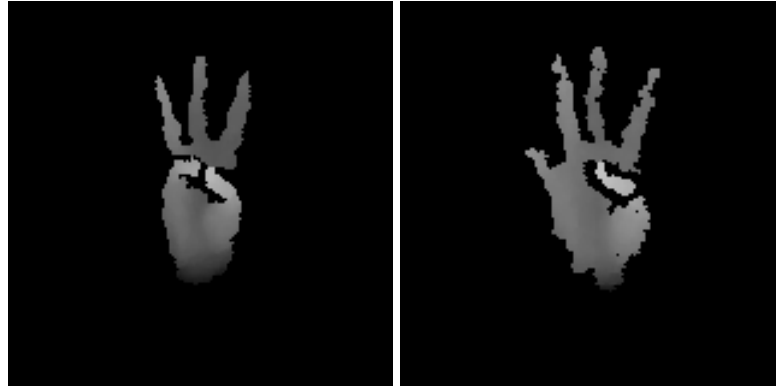


Figure 2.1.1: Illustration of the Kinect depth image noise and artefacts. It can be seen that the edges of objects are jagged, such as the edge of the hand in the two images. Also, there are discontinuities on the surface of objects caused by the “shadows” of objects closer to the sensor being projected onto the more distant objects. These shadow effects can be seen on the palm of the hand, where the little finger and thumb cause parts of the palm to become unrecognised by the sensor.

Noise and Artefact Removal

The data retrieved from the Kinect is very noisy and contains various artefacts. These artefacts include discontinuities in surfaces, jagged outlines of objects and shadowing effects when one object is in front of another. Furthermore, a large portion of the artefacts is not persistent across frames, occurring at irregular intervals.

We can effectively remove a large portion of the noise and artefact problems by simply averaging a set number of sequential frames. This works well because of the irregular nature of these distortions, but does introduce ghosting problems, where object movements leave behind a ghost image. This does however affect our system to a lesser extent, since we only focus on static hand poses. We found during testing that most of the noise is removed using this technique, despite its simplicity. Further investigation is however needed to determine if more complex techniques can produce higher quality filters.

2.1.3 Camera Calibration

The reader familiar with commonly used Computer Vision techniques might have noticed that we did not perform any camera calibration on either the depth or colour images. Camera calibration was not performed for several reasons as described below.

Firstly, we expect little to no distortion of the hand depth images. Both the depth and colour images retrieved from the Kinect do not have any noticeable distortions. Furthermore, we also assume that the user is located near the centre of the image, where camera distortion is normally less noticeable. The distortion on the hands will

be especially small, since the hand covers only a small area of the image, which is not larger than 160x160 for our system.

Secondly, we are not interested in precise world data for pose classification. Camera pose estimation or 3D reconstruction systems typically need very accurate world data to return an accurate result, thus requiring camera calibration. Our system is designed to be invariant to the form of the user's hand, because of the shape variations found between different users. We thus expect no loss in accuracy for small distortions which preserve the overall shape of the hand. The noise described in the previous section does in fact have a larger impact on the final performance of the system, thus more resources should be directed to using more complex filters than removing camera distortion.

Finally, we gain a small performance increase by not performing camera calibration, which is important considering our goal of a real-time system. This performance gain might be negligible for general desktop use since fast functions already exist, but this gain becomes more important in the scenario where the system is implemented on a mobile platform. We would still recommend future investigations into determining the true effect of camera calibration on the system.

2.2 Hand Tracking

This section discusses the structure of the hand tracking system implemented for this project. The system makes use of external libraries for hand tracking. Two libraries were considered for implementation of the hand tracker, namely OpenNI NITE and Skeltrack. The two libraries are evaluated based on their ease of use and hand tracking capabilities, with the OpenNI library performing better overall.

2.2.1 OpenNI NITE

OpenNI NITE is a closed-source middleware library which contains various algorithms for analysing the scene perceived by the depth sensor, including background segmentation and body tracking. NITE benefits from continuous support, which makes it suitable for developing production software. We used version 1.5 of the library, with the latest version being 2.2 at the time of writing.

We used the body tracking capabilities of the library in order to retrieve the location of the hands. The library only needs to be supplied a depth device as input and performs all the processing needed on the image stream to track the user's body.

The library can be used to extract the image coordinates of various joints from a user, including the coordinates of the hands. Furthermore, it can track multiple users simultaneously, though our testing system only tracks one person at a time, namely the first one to enter the scene. The NITE middleware library is however not open-sourced and the specific implementation of the body tracking is unknown.

The library needs a small amount of time for calibration before tracking the user. The user does not need to assume a specific pose during this time, though the calibration process is faster if the user is in full view with his arms a small distance away from his sides. The algorithm can retrieve a large selection of joints from the user's body once the calibration process is completed, including the user's hands. The hand tracking is accurate and fast, providing a good basis for the pose recognition system. The paper of [37] measured how accurate the library was able to track a user's hands by determining the distance between the tracked joints and the ground truth across a set of sequential frames. They found that the distance between the tracked joints and the ground truth was within 37 pixels for 90% of the tested frames.

2.2.2 Skeltrack

Skeltrack [38] is another body tracking library investigated during the course of this project. It provides an open-source solution for tracking the user's body joints. The Skeltrack library focuses on tracking the upper body of the user, including the head, torso and arms, unlike the NITE library, which does full body tracking. It is up to the developer to implement a system to track multiple users, since the library assumes the given depth image contains only one person. Furthermore, the version of the library tested assumed that only the pixels representing the user were present in the depth image, though this restriction has been removed in later versions.

The Skeltrack hand tracker is able to track the user's hand, but is not as robust as the NITE libraries. Partial occlusions of joints prove to be particularly problematic for the library, since it can cause inaccurate tracking of both the occluded and the non-occluded joints. This proves problematic in scenarios when the user's hands are occluding the arm and elbow joints, such as when the hands are held outstretched in front of the user. The library is also more resource intensive and not quite as responsive as the NITE libraries. These disadvantages motivated the choice to rather use the NITE libraries for hand tracking.

2.2.3 Implementation

The final implementation of the hand tracker used the NITE libraries to track the hand of a single user, which are configured to use the Kinect sensor. We retrieve only the hand, arm and head joints from the library. Our implementation only extracted the pixels of the right hand for testing using the procedure described below, but the same procedure can be applied for the left hand.

The hand pixels can be extracted by defining a bounding box around the hand and removing all pixels that do not fall within this bounding box, similar to the technique used by [39]. This solution assumes that the hand coordinates are accurate and located at the centre of the palm region. The width and height of the box are equal to the expected dimensions of the input image for the pose recognition component of the system, in our case 160x160. The depth of the box is equal to two times the average length of a hand, in our case chosen as 400mm. The depth value of the pixels that fall outside the box is set to zero in order to mark them as background pixels.

A region of interest can be extracted from the image once the hand pixels have been extracted. The region of interest is stored in a 160x160 depth image representing the hand, which is passed on to the pose classifier for classification.

Measuring the accuracy of the hand extractor requires labelled depth images where the body of the user is in full view. Retrieving and labelling the real-world depth images by hand similar to [2] can be time consuming. The process can be shortened by generating the test images with labels using a technique similar to the one described in **Chapter 4**, but requires the modelling and animation of a full body.

2.3 Pose Recognition

This section gives an overview of the complete proposed Pose Recognition System. It firstly discusses the problem of pose recognition from depth images and compares two solutions: pixel-based and feature-based classifiers. The section also documents the expected inputs and outputs of each individual component of the system and further discusses the various constraints and assumptions relevant to the input and output data for each system component.

2.3.1 Pose Recognition Problem and Solution

There are two approaches to solving the pose recognition problem. The first is a pixel-based approach, which uses the individual pixels of an image directly for pose classification. The second approach is feature-based, which first extracts important information from an image and uses the acquired information to classify the pose. These two approaches are briefly discussed in this section.

Pixel-based Pose Classification

It is possible to use the depth or classified region images directly for pose classification, by packing the pixel columns into an image vector and using these vectors for training and classification. It is then left to the training system to determine the important relations and features in the image for classification. There are however two problems with this solution.

The first problem is the exceptionally large number of training images needed to fully train an accurate pose classifier. This problem is caused by the fact that the image vector is not invariant to translations of the hand in the image or global depth offsets. This can be problematic if only a limited set of training data is available, however it is not a concern for our system since we can generate a large number of training images using a 3D modelling program.

The second problem however does concern our system, as it is related to the classification speed of a trained classifier. Accurate pixel-based classifiers are slower than similarly accurate feature-based classifiers. This is due to the number of dimensions an image vector contains and also the complexity of an accurate pixel-based classifier. This accuracy-speed trade-off is a large concern for our system, since our system needs to perform accurately and in real-time.

Feature-based Pose Classification

Keskin's [7] solution to the pose recognition problem and the one we use consists of three major steps and uses a feature-based approach. The first step is to process the raw depth image and label each individual pixel to a region of the hand. This labelled image is used to extract joint features, features representing the skeleton of the user's hand, for further processing. The final step extracts a further subset of features from the joints and uses the subset to classify the hand pose.

Speed is the main reason for using the joint extraction approach over directly classifying the depth image, as our system needs to perform accurately in real-time. The

proposed solution eliminates less relevant data from the depth image by extracting the hand joints and using the joints for pose classification. The joints can be extracted in real-time using a combination of Decision Trees and the Mean-Shift Algorithm to classify the depth image and extract the joints from the labelled image. The final joint vector has either 42 or 63 dimensions, depending on whether 2D or 3D joints are used for pose classification. Packing a depth image directly in a feature vector results in a vector with 160x160 dimensions, which can be costly to classify.

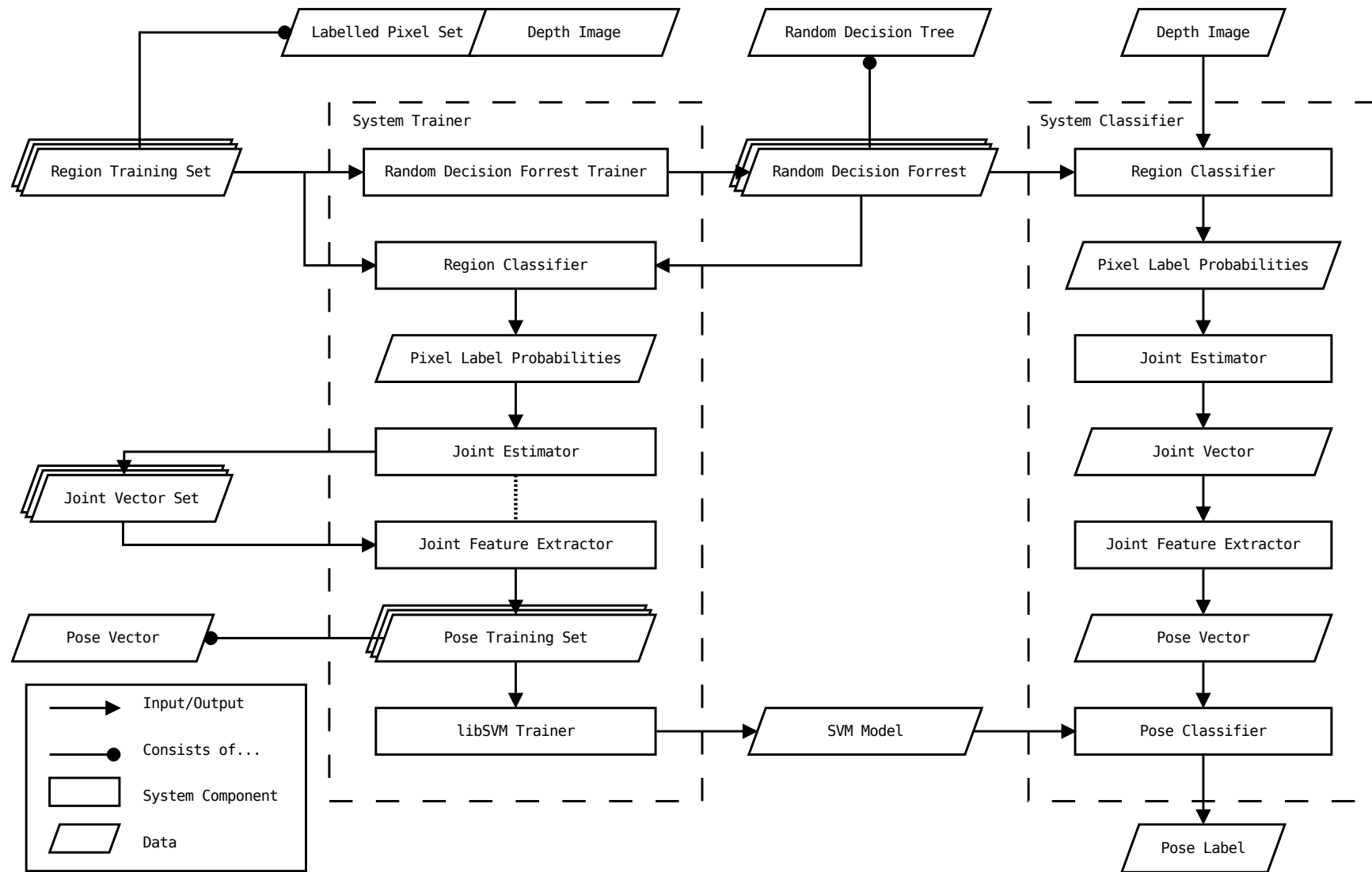


Figure 2.3.1: Overview of the complete pose recognition system.

2.3.2 Proposed System

The pose classification process can be divided into four phases: hand region classification, joint estimation, joint feature extraction and pose classification. A separate system component was designed for each of these steps and listed in this section. **Figure 2.3.1** shows the flow diagram of the complete system.

Region Classifier

The Region Classifier (RC) is responsible for calculating the label probabilities of the individual pixels of a depth image. The RC assumes that the 32bit floating point depth image given as input consists of only the hand and wrist, with the rest of the scene marked as background using the reserved value of zero (0.0). It further assumes that all of the hand and wrist pixels are contained within the 160x160 image and not cropped at the edges. The depth values of the input image are stored using metres as the measuring unit. The individual pixels are then processed using Decision Trees, discussed in **Chapter 3**.

The RC gives as output the labelled pixel set. Each labelled pixel consist of the pixel coordinates and a set of probabilities. The probability sets give the probability that a pixel belongs to one of the 21 labelled hand regions. The pixel probabilities are passed onto the Joint Estimator for further processing. The system implementation of the Region Classifier is discussed in more detail in **Chapter 4**.

Joint Estimator

The Joint Estimator (JE) extracts the joint coordinates from the labelled pixels retrieved using the Region Classifier. One of three algorithms are used to extract the joints: centre of gravity (COG), Mean Shift and our own modification of the mean shift algorithm called Joint Reservation. The COG and base Mean Shift algorithms evaluate the probabilities of each hand region separately and in one step, while the Joint Reservation algorithm uses an iterative process which places joints based on a joint confidence value and the placement of joints in previous iterations.

The joint coordinates extracted consists of the 2D image coordinates at which the joint is located and the depth value of the pixel represented by these coordinates. The coordinates are stored in a vector, which is passed on to the Joint Feature Extractor. The joint estimation process is discussed in **Chapter 5**.

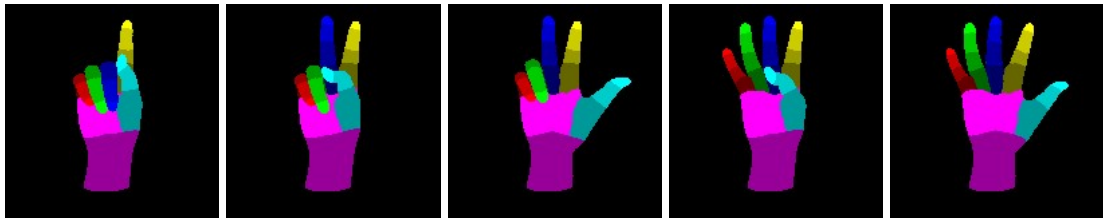


Figure 2.3.2: The ASL numerical digits (from left to right): 1, 2, 3, 4 and 5.

Joint Feature Extractor

The Joint Feature Extractor takes a joint vector as input, where each element consists of the joint coordinates as discussed above. The Feature Extractor processes this vector into a feature vector suitable for the final pose classification. The feature vector can also be invariant to certain transformations, depending on the type used. Three different feature types were tested for this project, where each type had a 2D and a 3D variation. The size and form of the final feature vector is dependent on the type of feature extracted, which also determines the pose classification model used to classify the feature. The Joint Feature Extractor is discussed in **Chapter 7**.

Pose Classifier

The Pose Classifier is responsible for the final pose classification. The Pose Classifier receives as input a feature vector and uses an SVM model trained using libSVM to classify the vector. The final output is a simple unique integer used to identify the pose represented in the depth image. SVMs and the implementation of the pose classifier are discussed in **Chapter 7**.

2.3.3 Recognised Poses

The system of Keskin [7] was able to classify 10 different American Sign Language (ASL) digits with a recognition rate of 99.9% using real world data retrieved from the Kinect. The paper of [7] does not however specify which digits were used, making it unclear whether the system will perform well using digits which have a similar hand pose.

We decided to train and test our system using 12 different ASL digits. The ASL digit set was chosen because it is common and well known, making it easier to compare our results with other systems which also use the ASL digit set. A limited set of digits was used to make our results comparable with that of [7] who used ten digits.

The first five digits are simply the numerical digits 1 through to 5 (**Figure 2.3.2**),



Figure 2.3.3: The ASL alphabetical digits (from left to right): A, S, L, Y and W.

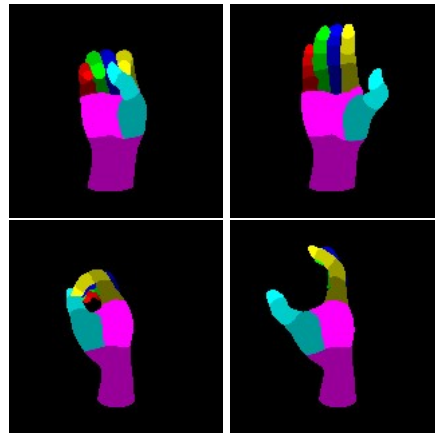


Figure 2.3.4: The ASL alphabetical digits O (left) and C (right). The top row shows the front view of the digits, while the bottom row shows the side view.

while the next five digits are the alphabetical digits A, S, L, Y and W (**Figure 2.3.3**). These 10 digits test whether the system can differentiate between fully extended and fully contracted fingers and whether the pose classifier can distinguish between the five fingers.

We also added the alphabetical digits O and C (**Figure 2.3.4**), where the fingers are only partially extended or contracted. The depth images of these two digits will thus have a larger range of depth values, as opposed to the previous 10 digits where the visible hand surface lies on roughly the same plane.

The 12 digits mentioned above can be classified by only determining how much each finger is extended relative to the palm. A system designed for these digits will thus not be able to distinguish between digits where contact between fingers is also a factor in classifying different digits. We decided to create two variations of the digits 2, 3, 4, 5 and W to test whether our system could differentiate between “open” and “closed” poses (**Figure 2.3.5**). The “open” variations are characterised by extended fingers which do not come into contact with other fingers, while the extended fingers of the “closed” variations do come into contact. The final set of 17 poses used to evaluate our system is shown in **Table A.1.1**. Using this small set of poses makes our system comparable to the system of [7] which used 10 poses.

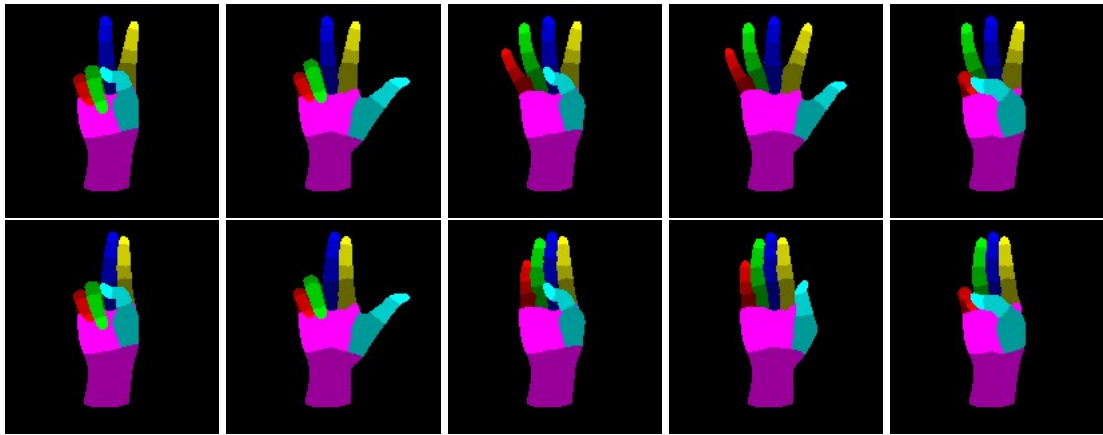


Figure 2.3.5: The “open” (top) and “closed” (bottom) variations of the ASL digits (from left to right): 2, 3, 4, 5 and W.

2.3.4 Input Data Constraints

This section will discuss the assumptions and corresponding constraints placed on the system input data for both the training and classification process. The three classes of constraints discussed relate to the physical pose and position of the hand as perceived by the sensor, the constraints placed on the training data and the constraints placed on unseen data used during testing.

Simplification is needed to create a practical system. Without these constraints, the system might not be accurate enough to be usable. We did not aim to solve the complex problem of sign language recognition which is still only partially solved. To the contrary, our system should be able to control a computer system which requires a much smaller set of poses to be recognised. Furthermore, some of the constraints potentially make the system more user friendly. since the user has to enter an input pose. The system will otherwise assume the user is not actively trying to communicate to it, thus filtering out unintentional commands. This feature is however not fully implemented in our current system, since it requires either an “unknown” class or pose label probabilities as output instead of just a label.

Physical Constraints

Two constraints were placed on the physical input given to the system as observed by the depth sensor. These constraints relate to the pose, orientation and position of the user’s hand relative to the depth sensor.

The first constraint is placed on the expected input poses. The system assumes that the hand is in one of the predefined poses and is unable to differentiate between a known and an unknown pose. The system is thus not suitable for pose classification

purposes in the scenario that unknown poses are given as input, but could still work well as a hand joint estimator.

The second constraint relates to the orientation and position of the hand relative to the depth sensor. The data set used to train the system contains images of which the hand rotation and position relative to the depth sensor are constrained. It is preferable that the given input should be within these constraints for the system to perform optimally. The system should thus be trained with a new set of orientation and position constraints if the expected input changes at a later stage. These two constraints apply to both the training data and the unclassified input data, as given to the Region Classifier.

Training Data

The proposed system uses supervised trainers for building both the Region and Pose classifiers. The generation of synthetic training data makes it possible to automatically label the training data during the generation process.

The Region Classifier is trained using images which store both depth and hand region data. The measuring units of the depth values are in metres. The depth value zero is reserved for non-hand regions, such as the background or other parts of the user's body. The region labels are stored as RGB colours, where a unique colour is allocated to each region. The use of a colour value makes it possible to use a 3D modelling program to generate training images.

Our system's region classifier was trained using a simple model where only one unique colour is allocated to a region. On the other hand, shading of the region colour labels can be used to indicate the probability that a pixel belongs to a certain region, which can be useful for modelling the boundaries between regions.

The Pose Classifier is trained using either a subset of the Region Classifier training data or a separately generated training set. The pose labels are determined during the generation of the training set, similar to how the region labels are assigned.

Unclassified Data

Three assumptions are made with respect to the depth image given as input for classification. The assumptions relate to form of the data present in the input image and determines which types of images the system is expected to classify correctly.

The first assumption relates to the data present in the input depth image. It is assumed that the image contains only the pixels of the user's hand and partial wrist. The rest of the user's body and other background objects are represented with the reserved background depth value of zero. It is preferable that the wrist is partially present in the image, as it is used to determine the true location of the palm. The wrist should be cropped closely to the hand, to avoid incorrect classification of palm and finger pixels on the extended portion of the wrist.

Another assumption the system makes is that the hand is contained fully within the depth image and is not cropped at the edges. It is preferable that the image is centred around the palm of the user's hand, as some of the simpler tested features are not invariant to translations. Furthermore, the system assumes that the input image contains the user's right hand, as only right hand images were used for training. The hand tracker should determine which hand is sent to the Pose Recognition system and can simply mirror the image around the vertical axis as necessary to conform with this constraint.

Lastly, the system assumes that the input image contains only small amounts of noise. It assumes that the largest part of the hand surface is smooth and continuous. The fingers should also be distinguishable from each other and the palm when in close contact with each other. We found that by applying an average filter over 3 sequential depth frames, a noisy image retrieved from the Xbox Kinect could largely comply with this constraint.

2.4 Summary

This chapter gave an overview of the various elements used to create the Pose Classification system for this project. **Section 2.1** described important aspects regarding the depth sensor interface of the system. These aspects included OpenNI, the library used to interface with the Kinect, how the depth data is processed and why camera calibration is not needed for our system. **Section 2.2** discussed the Hand Tracking module of the system, describing the external libraries considered for use (OpenNI NITE and Skeltrack) and the implementation of the hand tracker. **Section 2.3** described the Pose Recognition module. The section described Keskin's [7] solution to the Pose Recognition problem, our proposed system based on Keskin's solution and the chosen set of poses our system is designed to recognise. The next chapter discusses the theory behind the Region Classifier, which is the first component of our system.

Chapter 3

Decision Trees

The first component of the pose recognition system is the Region Classifier (RC), which classifies the pixels of a depth image according to the region of the hand the pixels represent. The RC will be discussed in greater detail in **Chapter 4**. This chapter focuses on the theoretical component on which the RC is based, namely Decision Trees.

The first section introduces the concept of decision trees. **Section 3.2** describes the training process for decision trees and the design aspects to consider. **Section 3.3** discusses the combination of several decision trees into a forest to create a more accurate classifier. The final section describes the variation of decision trees used in this project, namely Random Decision Trees.

3.1 Definition

Decision trees are discussed in pattern recognition literature [40, 41] and have been used for various applications [2, 7, 42]. Decision trees are tree-based classifiers which store various tests in internal nodes and classification rules in leaf nodes. A data point is classified by propagating it through a tree, where the path of propagation is determined by the outcome of the various subsequent tests found along the path. The classification rule stored at the end of the path is used to classify the data point. Decision trees are used because of their speed and accurate classification for constrained problems, though the classifier can suffer from overfitting.

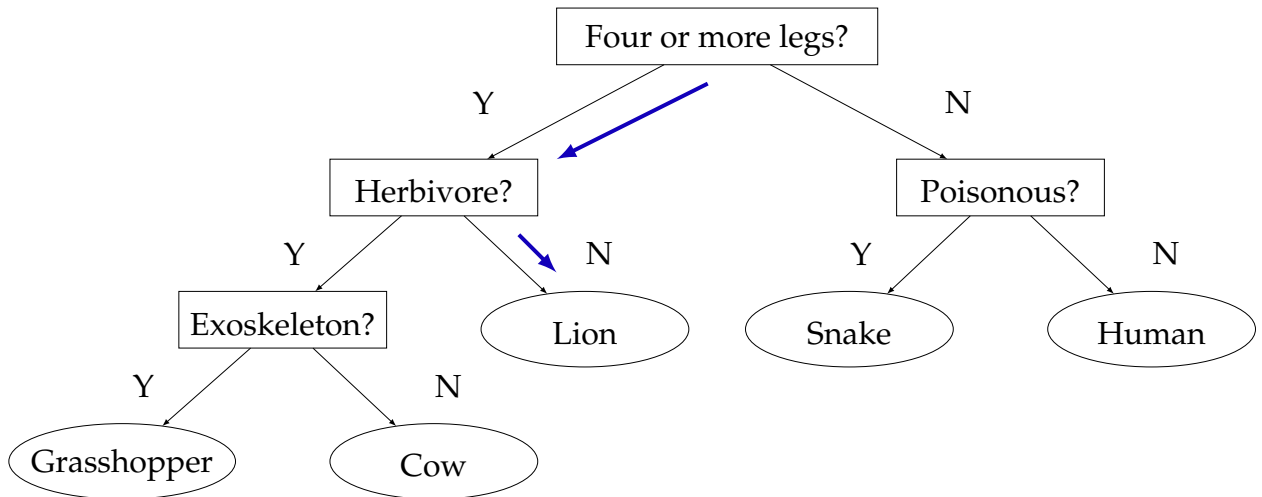


Figure 3.1.1: An example of a simple Decision Tree for identifying different animals.

Figure 3.1.1 illustrates a simple decision tree. The tree is used to classify one of five classes using the various tests stored in the internal nodes. The classes are:

[Grasshopper, Cow, Lion, Snake, Human]

As shown by the blue arrows in **Figure 3.1.1**, a data point is classified as belonging to the *Lion* class by determining:

1. It has four or more legs.
2. It is not a herbivore.

The set of tests of which a path consists is collectively known as a rule, where each rule corresponds with a specific leaf node. The leaf node stores the classification rule, which in the previous example is the assignment of a single label (*Lion*). Each of the five example classes can be classified using a separate rule from the decision tree's rule set. For example, a data point is labelled as *Cow* if it adheres to the rule:

1. Has four or more legs.
2. Is a herbivore.
3. Does not have an exoskeleton.

Only those tests that are applicable to a data point are used for classification of that data point, since the subsequent tests propagate the data point down a single path. This makes decision trees extremely efficient for classification problems. The following sections will discuss a few of the aspects which should be considered when designing and training a decision tree.

3.2 Training

This section discusses the different aspects to consider when training a decision tree. The training procedure is designed to find the set of splits which will minimise the classification error of the decision tree. A top-down approach is used to train the decision trees for this project, where the original training data is split into smaller parts further down the tree. We will thus focus only on the design considerations of designing a top-down trainer, as opposed to the bottom-up approach.

The first subsection gives an overview of the training procedure. The overview is followed by more detailed descriptions of important aspects of the training procedure and how they influence the final output of the training process. **Algorithm 1** shows the algorithm used to train the decision trees.

3.2.1 Training Procedure Overview

As previously mentioned, we use a top down approach to train the individual decision trees, which involves propagating the training data from the top to bottom of the tree. The complete training set is used to train the root node, from where it is split into subsets which are propagated down children nodes. All internal nodes are trained using the same procedure:

1. Training data is assigned to the current node from its parent.
2. The data is used to train an optimal test for the current node.
3. The optimal test is used to split the data into subsets.
4. The subsets are propagated down to the children for training.

The process is continued recursively until a stopping condition is met and a leaf node is created. The data subset assigned to a leaf node is used to create a classification rule which is stored by the leaf node. The training procedure is illustrated in **Figure 3.2.1**.

The training data is always processed in one batch. A finalised data set is therefore needed before training starts, since new data cannot be added once the process is started. The following sections discuss the various aspects that needs to be considered when designing a decision tree trainer.

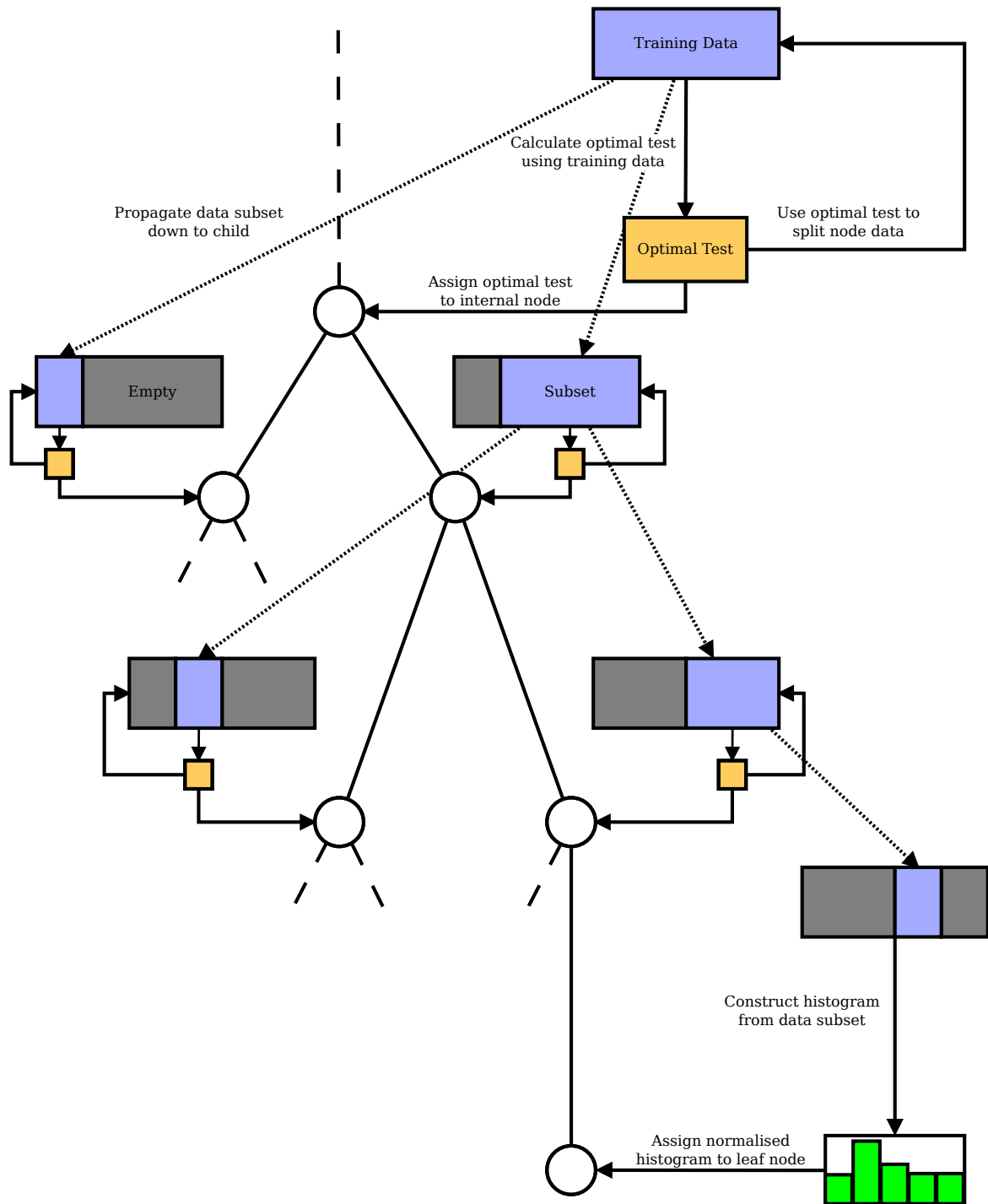


Figure 3.2.1: General training procedure for a decision tree.

3.2.2 Split Types

The first aspect to consider when designing a decision tree involves the choice of tests to store in the internal nodes. Decision trees can have multiple-outcome tests which can lead to more complex classifiers, potentially increasing the accuracy of the tree at the cost of speed. We shall focus on the binary case such as shown in the previous example for illustrative purposes. The tests will be referred to as splits, since the

tests determine how the data is split between children nodes during training. Three split types are discussed in [40] and mentioned in [13], namely Axis-Parallel Linear, Oblique Linear and Non-Linear Splits.

Axis-Parallel Linear Splits

This split type considers only one dimension of the feature space during a split. A hyperplane is placed perpendicular to one of the feature space axes at a point τ . The data is then split according to which side of the hyperplane the data resides. The equation for axis-parallel linear splits is:

$$x_i < \tau \quad (3.2.1)$$

where x_i is the feature value of the i^{th} dimension and τ is a threshold value. These splits essentially split the feature space into halves, where the one half corresponds to $x_i < \tau$ and the other half corresponds to $x_i \geq \tau$. This creates rectangular regions in the feature space for classification, which can perform poorly when classifying data with irregular distributions. These splits do however create decision trees which perform classification extremely fast, because only one dimension needs to be processed at a given split.

Oblique Linear Splits

This split type is a generalisation of the axis-parallel splits. It creates hyperplanes which can be oblique to the axes of the feature space. These splits have equations that are of the form

$$\sum_i a_i x_i < \tau \quad (3.2.2)$$

where x_i is the i^{th} feature value of the feature vector \mathbf{x} and a_i is the weight associated with the corresponding feature value. These tests are more accurate compared to axis-parallel test when classifying irregular data distributions at the cost of speed. The decision trees constructed for this project use tests trained using oblique linear splits as discussed in **Section 4.2.1**, because of their increased accuracy over Axis-Parallel splits and relatively low computational cost when compared to Non-Linear splits.

Non-Linear Splits

The third test type is based on a non-linear combination of features and is of the form

$$f(\mathbf{x}) > 0 \quad (3.2.3)$$

where $f(\mathbf{x})$ is a non-linear function using the individual features of feature set \mathbf{x} . These splits represent the most general form and are also computationally the most expensive (assuming a complex function is used).

3.2.3 Split Criterion

As discussed in **Section 3.2.1** an internal node is trained using a subset of the training data. The final test assigned to a node will determine how this data is split and assigned to train the node's children. There is a large number of possible tests that can be used to split the data between the nodes. Choosing which one of these possible tests to assign to a node is however not trivial. A rule is thus needed to evaluate all possible tests, which would make it possible to choose an optimal test from the set of tests. This rule is known as the split criterion.

Split criterion refers to the criteria used to decide which one of several candidate tests are to be assigned to a given internal node when constructing a decision tree. This will ultimately affect the shape of the final decision tree, which has an influence on the speed and accuracy of the tree. The rest of this section will illustrate the effect the tree shape has on classification. It will also describe the split criterion used for this project in order to optimise the shape of the trained decision trees.

Tree Shape

Examining the rules of the decision tree in **Figure 3.1.1**, it should be evident that the order in which the tests are performed does not have an effect on the classification results. It does however affect the average length of a rule, which directly affects the computation time required to classify a set of data points. The time complexity of a searching operation for a binary decision tree is $O(d)$, where d is the depth of the tree. Balancing the binary decision tree will also minimise the tree depth such that $d = \lfloor \log_2(N) \rfloor$, where N is the number of nodes in the tree. It is thus preferable to implement a classifier with a balanced decision tree for fast classification.

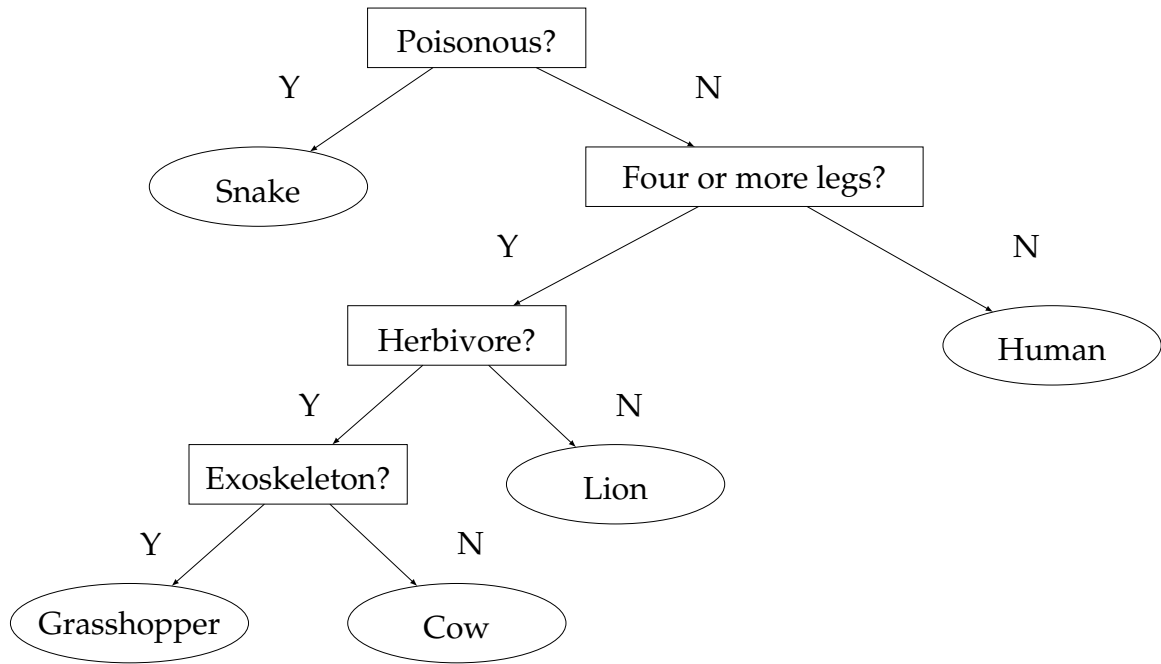


Figure 3.2.2: The modified decision tree.

Table of Rule Lengths		
Rule	Original Tree	Modified Tree
Grasshopper	3	4
Cow	3	4
Lion	2	3
Snake	2	1
Human	2	2
Average	2.4	2.8

Table 3.2.1: The average length of a rule for the original and the modified decision trees shown in Figure 3.1.1 and Figure 3.2.2.

Consider the modified tree shown in **Figure 3.2.2**. The same set of tests are performed to evaluate a set of data points, but the order of the questions have been altered slightly. **Table 3.2.1** shows a comparison between the original tree's and the modified tree's rule lengths. As can be seen, the modified tree will on average need more calculations to classify a data set with a uniform spread of points across the five classes. Further note that the modified tree needs four levels of internal nodes to correctly classify all the classes, while the original only needs three. If we were to trim the modified tree to use only three levels of internal nodes for classification, it would not be able to distinguish between the *Grasshopper* and *Cow* classes.

Shannon Entropy

The previous example shows that it is not only important to choose an adequate set of tests but to carefully order these tests in such a way that the tree depth is kept at a minimum. Arguably the most intuitive split criterion involves simply choosing the test which spreads the classes most evenly among the children of the node being considered. This is again illustrated by the examples from **Figure 3.1.1** and **Figure 3.2.2**. The root node of the tree from **Figure 3.1.1** uses a test that splits the labels more evenly between the two children, which creates a tree which is eventually shallower than the tree from **Figure 3.2.2**. This approach can however prove problematic when a more complex data set is used, where it might not always be possible to find a split which divides the labels evenly between children nodes.

The split criterion used by [42], [2] and [7] involves choosing the split which maximises the decrease in entropy between the original data set and the two subsets. The decrease in entropy is calculated as

$$f_{split}(S_P, \phi) = H(S_P) - \sum_{s \in L, R} \frac{|S_s(\phi)|}{|S_P|} H(S_s(\phi)) \quad (3.2.4)$$

where $|\cdot|$ denotes the total number of samples in a set, ϕ denotes the split rule used to split the data and S_P is the original data set with the two subsets S_L and S_R . $H(S)$ is the Shannon Entropy calculated for the set S :

$$H(S) = - \sum_{c=1}^n p_c \log_2(p_c) \quad (3.2.5)$$

where p_c is the probability of class c within the set S . The split with the highest score according to (3.2.4) is chosen when splitting a node during training. It can be seen that with each successive data split, the data spread over the various classes will be more concentrated around a smaller subset of classes than for previous data splits. Each branch of a tree will thus specialise on a different set of classes for classification. This split criterion also balances the decision tree to a certain extent, which improves the classification speed of the final classifier.

3.2.4 Classification Rule

The leaves of the example shown in **Figure 3.1.1** simply store class labels for classification. This is however not adequate for real world data, where the different class distributions often overlap and cannot easily be separated. **Figure 3.2.3** shows the flattened feature space of the decision tree shown in **Figure 3.1.1**, where the feature space

is divided into rectangular regions by the decision tree. If the tree from **Figure 3.1.1** was used to classify the data from **Table 3.2.2**, it would incorrectly classify the data point belonging to the *Cat* class as *Lion*.

This problem is solved by rather storing a set of class probabilities in each leaf node instead of a single class label. These probabilities are generally stored as a histogram of class samples. Given a decision tree with a set of predefined rules, we can train the class labels by classifying a training data set. Each time a data point reaches a leaf node, the histogram bin of that leaf which is associated with that data point's class is incremented. Once the training data set has been classified, we can normalise the histograms to retrieve the class probabilities. These class probabilities can then be used in further calculations to estimate the class label of a data point, even if the testing set contains overlapping class distributions.

Using the data from **Table 3.2.2** to train the decision tree of **Figure 3.1.1**, the leaf which originally stored the *Lion* label will be changed to store the following class probabilities:

Class	Grasshopper	Cow	Cat	Lion	Snake	Human
Probability	0.0	0.0	0.25	0.75	0.0	0.0

The new class probabilities should give a better estimate of the class of a data point which adheres to the rule of this leaf.

Note that another solution to the problem above is to add another internal node to extend the rule for *Lion* to also include the test *Large?*, which would allow the tree to differentiate between *Cat* and *Lion*. This does however increase the complexity of the classifier and increases the average depth of the tree, which can lead to a slower classification time for certain data sets. It also does not guarantee a more accurate classifier, since the class distributions of a data set may overlap (exceptionally large cats and small lions).

3.2.5 Stop Rule

The final consideration when designing the training process for a decision tree concerns the stopping conditions related to node creation. The set of conditions that need to be met before creating a leaf node is known as the Stop Rule. The most basic stopping rule simply states that a leaf node must be created when the subset of data prop-

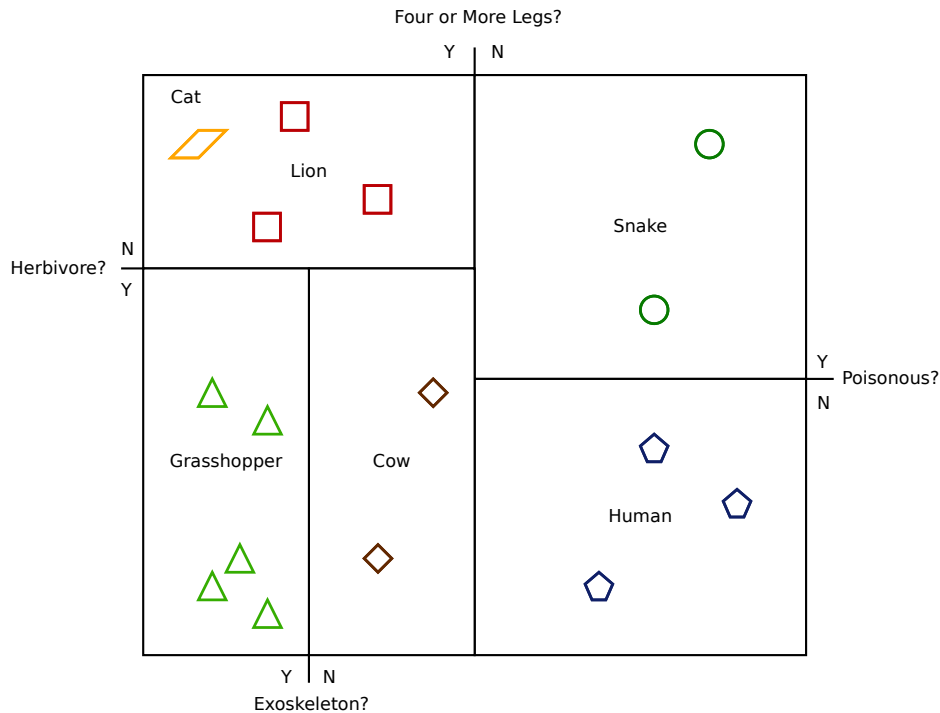


Figure 3.2.3: Illustration of how the Decision Tree from **Figure 3.1.1** splits the feature space (flattened to two dimensions).

Table of Class Samples	
Class	Number of Data Points
Grasshopper	5
Cow	2
Lion	3
Cat	1
Snake	2
Human	3

Table 3.2.2: The number of data points belonging to each class.

agated to the node consists of only one class and cannot be split any further. This rule will produce a pure leaf node of which the classification rule consists of a single class label. This stop rule gives the best results when testing the classifier on the training set, but will generally perform poor when tested on other data.

The basic stop rule can be improved with two simple additions which not only help to slightly reduce overfitting, but can also decrease the training time of a decision tree by limiting the number of nodes that needs to be trained.

Depth Limit

A simple addition that can be made to this rule is to limit the maximum depth of the decision tree. If a depth limit appropriate to the training data set is chosen, it should

limit the amount of pure nodes while still allowing the tree to differentiate enough between classes to perform accurately.

Entropy Decrease Threshold

The above addition does still leave room for shallow branches to suffer from overfitting. A further addition which focuses on each individual node instead of on a specific depth involves adding an entropy decrease threshold. The value calculated from (3.2.4) can be used to get an indication of how much information can still be extracted from a data subset used to train a node. If this value falls below a given threshold, it can be assumed that there is not any information left to be gained from the subset and that further splitting will result in overfitting. It is thus beneficial to stop the splitting process at this point and assign classification rules to the children based on the current data split.

3.3 Decision Forests

Overfitting is a constant concern when training decision trees, because of the inherent nature of the training procedure to create a tree which specialises to the training data. **Section 3.2.5** showed how making a few simple modifications to the basic stop rule can help to reduce this problem, but still leaves a lot of room for improvement. The papers of [2] and [7] used a number of decision trees combined into a forest to compensate for the overfitting of individual trees, which will be discussed in this section.

The paper of [12] introduced the notion of training multiple decision trees using different subspaces of the same training set and combining them into one classifier. It is argued that each tree should specialise differently to the training set, thus averaging the classification results of all the trees will give a more generalised result. The author provided results which illustrated how adding more trees to a forest increased the accuracy of the combined classifier when tested on a test set, while still performing well on the training set.

The data point \mathbf{x} can be classified using decision tree D_k to retrieve the set of M class probabilities \mathbf{p}

$$D_k(\mathbf{x}) = \mathbf{p}_k \quad (3.3.1)$$

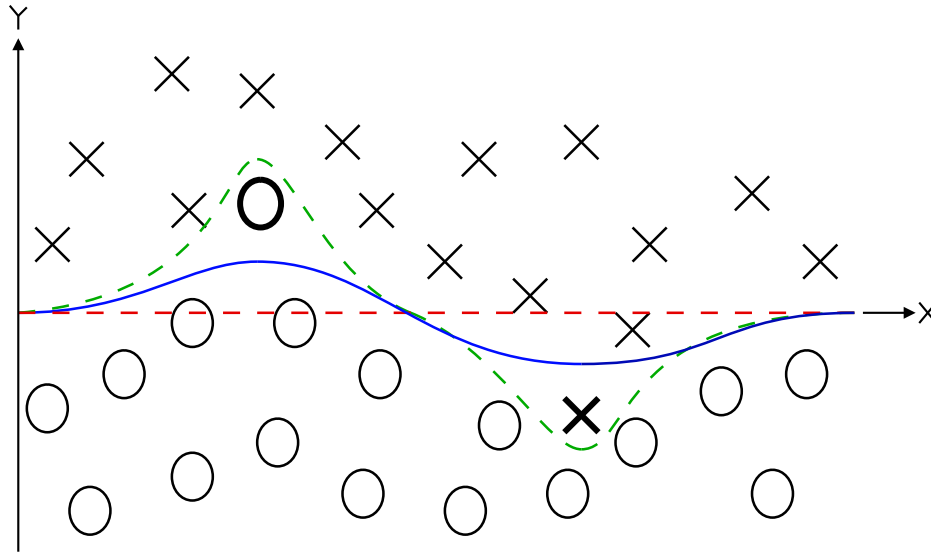


Figure 3.3.1: An example problem showing how a combined classifier can prevent the effects of overfitting. The two outliers of the circle and cross classes are marked in bold. The green and red striped lines show the decision boundaries of the two classifiers trained using different subspaces of the data set (XY- and Y-subspaces respectively). The solid blue line shows the combined classifier found by averaging the previous classifiers.

where

$$\sum_{i=1}^M p_i = 1 \quad (3.3.2)$$

Combining N_F decision trees into a single forest classifier D_F and using the resulting classifier to classify \mathbf{x} give

$$\begin{aligned} D_F(\mathbf{x}) &= \frac{1}{N_F} \sum_{i=1}^M D_i(\mathbf{x}) \\ &= \frac{1}{N_F} \sum_{i=1}^M \mathbf{p}_i \end{aligned} \quad (3.3.3)$$

The concept of multiple classifiers is illustrated with a simple example problem in **Figure 3.3.1**. It shows a hypothetical data set consisting of two classes represented with circles and crosses. The data set contains two outliers, one from each class, which are marked in bold. The decision boundaries of two separate classifiers are shown, as well as the decision boundary of a combined classifier.

The striped green line represents the decision boundary of a classifier trained using the full feature space. Assuming the classifier was trained to have a perfect recognition rate when tested with the training data, as is the case with decision trees, it would

suffer from overfitting as shown. It should be evident that adding a small amount of noise to the set could easily cause points around the outliers to be incorrectly classified.

The striped red line represents the decision boundary of a classifier trained using a projection of the data set on the Y-axis. The classifier will perform well when tested on the training set, but similar to the first classifier, will suffer from overfitting problems when adding noise to the set.

The solid blue line represents a combined classifier that in this case simply averages the previous two classifiers. The performance of this classifier should be less affected by noise, considering the decision boundary is better spaced between the classes. This averaging of multiple classifiers proved to be fast and effective in our tests. Further investigation is required to determine if there is better solutions.

The specific implementation of decision forests in this project is discussed further in **Chapter 4**. It should be noted that if individual classifiers do have similar specialisation characteristics, that a combined classifier would still suffer from the effects of overfitting. The next section will discuss a technique which uses randomisation to ensure that individual decision trees trained from the same training set are unique.

3.4 Random Decision Trees

Section 3.2.3 discussed how the best test from a group of candidate tests can be selected during the training of a node using Shannon Entropy. This section will describe how a set of these candidate tests can be generated using a variation of Decision Trees which uses randomisation, namely Random Decision Trees (RDT). RDTs have been used numerous times in literature for various classification problems [2, 7, 42, 12, 13].

The first step after assigning training data to a node is to find an optimal test for splitting the data. The randomisation algorithm used to build RDTs generates a set of candidate tests, from which an optimal test is chosen. Randomisation provides a fast way of exploring the feature space of the problem. Various constraints can be placed on the randomisation algorithm to explore only the applicable subsets of a feature space, which can improve the efficiency of the algorithm considerably.

One of the benefits of randomly selecting test candidates is that multiple unique trees can be generated from the same training set. This attribute is advantageous when training forest classifiers with a limited training set, since it is expected that each tree will use candidate sets unique to that tree and thus specialising differently to the train-

ing data than other trees. This is needed to create effective forest classifiers as discussed in the previous section.

RDTs are used in this project because they can be used to quickly explore the feature space of a given problem. The ability to generate unique trees from the same training set is not of particular interest for this project, as **Section 4.1** will show that the amount of training data available is effectively infinite. Each tree will be unique, since it is possible to assign a unique training set to train each RDT.

3.5 Summary

This chapter discussed the theory behind the Region Classifier created for this project. In **Section 3.1** and **Section 3.2**, we discussed the definition and training of Decision Trees. In **Section 3.3**, we described how a set of unique Decision Trees can be combined into a Decision Forest to create a more general classifier that performs better on testing data. **Section 3.4** described Random Decision Trees, which is created using a randomisation training algorithm which efficiently explores the feature space of the training data. The next chapter describes how these concepts are combined to create the Region Classifier for this project.

Algorithm 1 The recursive algorithm used to train one binary Random Decision Tree.

function TRAINNODE(Node, Training_Data, Stop_Recursion)

if Stop_Recursion is False **then**

for N_GENERATED_SPLITS **do**

 Generate random Split

while Split is in Split_Set **do**

 Generate new random Split

 Add Split to Split_Set

for Split in Split_Set **do**

 Use Split to split Training_Data into sets A and B

 Calculate Shannon entropy Score of sets A and B

 Store Score in SE_Scores

 Best_Score \leftarrow First Score in SE_Scores

 Best_Split \leftarrow First Split in Split_Set

for (Score, Split) in (SE_Scores, Split_Set) **do**

if Best_Score > Score **then**

 Best_Score \leftarrow Score

 Best_Split \leftarrow Split

 Node.Split \leftarrow Best_Split

 Create Node.Left_Child & Node.Right_Child

 Use Best_Split to split Training_Data into sets A and B

if Best_Score > STOP_CONDITION **then**

 TrainNode(Node.Left_Child, A, False)

 TrainNode(Node.Right_Child, B, False)

else

 TrainNode(Node.Left_Child, A, True)

 TrainNode(Node.Right_Child, B, True)

else

 Create normalised histogram Probabilities from Training_Data

 Node.Probs \leftarrow Probabilities

Chapter 4

Region Classifier

The Region Classifier (RC) is the first component in the pose recognition system. The RC is used to classify the depth images given as input to the system. Each pixel of the depth image is matched to the corresponding hand region, which can then be used to estimate the hand joint positions. The joint estimation is discussed in **Chapter 5**.

We chose Random Decision Forests (RDF) to implement the per pixel Region Classifier (RC), because the classifier performs fast and accurate with constrained data. Furthermore, classification takes place in real-time which is important for our application. The papers of [7] and [2] implemented object pose recognition using RDFs. Both papers showed that RDFs are accurate and can perform in real-time. They further showed how the Mean-Shift Algorithm could be used to estimate the joints from labelled depth images, which is discussed in **Chapter 5**.

The previous chapter discussed the theory behind decision trees, showing how they can be used as classifiers and also how RDFs can be used to increase the accuracy of these classifiers. This chapter will show how the RDTs were implemented to classify various hand regions from a depth image. It will illustrate the depth features used and the generation of synthetic images for training and testing purposes.

4.1 Generating Synthetic Images

One important aspect of the techniques proposed by [7] and [2] is the generation of synthetic depth images for training purposes. There are many advantages to using synthetic images for training, including the possibility to create an infinitely large data set with infinite variation. Image filters can also be applied to the rendered images to simulate camera distortion or noise, in order to train classifiers that are resistant to

these effects. This section will describe the tools and parameters used to generate these images.

4.1.1 Rendering Application

The synthetic images used to generate the training and test sets for this project were rendered using Blender (www.blender.org). Blender is a freely available open source 3D modelling and animation program.

Blender was chosen over other applications for various reasons. Firstly, it is well documented and has a large amount of tutorials available online which accelerated learning the basic usage of the program. Secondly, it is one of the standard applications used for modelling and many professional Blender models are available freely online for academic and personal use. Using one of these models removes the need to create a new model from scratch, which can be time consuming and requires some skill to do correctly. Finally, Blender has a scripting facility that utilises Python code, which can be used to animate a scene and move the camera. This facility is especially important, since it allows to automate the process of creating pose and viewing angle variations.

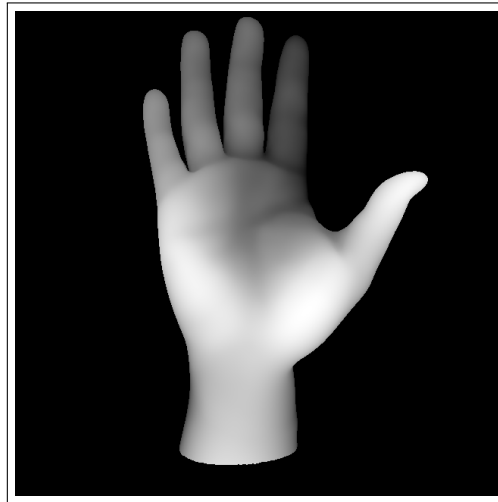
4.1.2 3D Hand Model

The hand model used was retrieved from the LibHand library [43], which is free to use for academic purposes under the *Creative Commons Attribution 3.0 Unported License*. Three important adjustments were made to the original Blender model of LibHand.

Firstly, the skin texture of the hand was modified as shown in **Figure 4.1.1**. Each of the hand regions is represented by a unique colour, which the training application can map to a region label using a nearest neighbour algorithm. These region labels are used during training and testing. Shading and texture filtering was turned off to keep the colours assigned to the regions constant and reduce errors in the colour to label mapping.

The second modification to the LibHand Blender model was the addition of a pose library, which contains the various poses which the classifier must be able to identify. The library was used to return a pose to its original state after rendering a variation of the pose.

The third modification to the Blender model was changing the measurement units in such a way that the hand had a length of 200mm. This ensures the measurement

*(a) Labelled Hand**(b) Depth Image**Figure 4.1.1: Example of a labelled and a depth image.*

unit of the generated depth image is in millimetres, the same as that of the Kinect.

Other modifications included moving the centre metacarpal bone to the origin of the scene and changing a few of the labels assigned to various bones in the model to improve readability of the python script.

4.1.3 Image Generation

Blender provides a Python scripting environment, which makes it possible to make various changes to the various objects in a scene. The project uses this environment to generate the various poses and camera views needed and to apply them to each scene before rendering the scene. A python script was written and added to the Blender

model to automate a few processes.

As discussed in the previous chapter, RDTs tend to suffer from over-fitting. This was partially avoided by generating a number of variations of each of the poses in the pose library. Each variation had small random rotations applied to the joints in such a way that each variation pose was unique. A slight 3D offset was also applied to the position of the hand with respect to the scene origin for each image rendered.

Images of each variation were generated from a number of different camera angles and distances. The set of camera poses were chosen in such a way as to correspond to the set of views we expect the system to see in practice and still recognise the user's commands. The constraints that were placed on the system during testing are discussed in **Section 4.3.1**, while the parameters used for the training and test sets are discussed in the next section.

The generated images were exported as EXR images using the OpenEXR library (www.openexr.com). The EXR format allows to store high dynamic range images, which is needed for the depth image. The output images consisted of the three 8-bit RGB channels and a 32-bit depth channels. The RGB channels store the image labels as colours while the depth channels store the depth from the camera in metres.

4.1.4 Generation of Training and Test Sets

Two large sets of images were generated for training and testing purposes. Different constraints were placed on the two sets in order to better determine the characteristics of the region classifiers trained using the design described in this chapter. **Table 4.3.2** lists the constraints placed on the two sets.

A base variation of each of the 17 hand poses were modelled in Blender. The base variations are all set in an upright position, with the fingers to the top of the image and the wrist to the bottom. The rotation and scaling factor of these base variations are measured in Euler angles as $[0^\circ, 0^\circ, 0^\circ]$ and 1 respectively. Similarly, the rotation and scaling factor of the individual bones of these variations are measured as $[0^\circ, 0^\circ, 0^\circ]$ and 1 respectively.

During the generation process, a set number of variations of each pose is generated. Each variation uses a hand with a different size and shape, where the overall scaling determines the hand size and the individual joint scaling determines the hand

shape. This synthetic generation of different hand shapes gives a rough representation of what the different shapes might look like, although hand made variation models should be more representative.

After a variation's hand shape is determined, the pose shape is changed by slightly rotating the individual bones of the fingers. Small rotations are used to ensure the actual pose label does not change, which would lead to incorrect labelling of the training and testing data. These scale, shape and pose variations ensure that the final region classifier does not specialise towards a specific hand shape.

Position of User Hands

It is assumed that the user will interact with the system with his hands in front of him, his palms facing towards the camera. Training and testing images are generated accordingly, which allows for the hand to be rotated $[-30^\circ, 30^\circ]$ around the x and y and $[-60^\circ, 60^\circ]$ angles around the z axis, as shown in Figure 4.1.2. These constraints allow for smaller training data sets and should produce a system that has a higher accuracy. Furthermore, it promotes a natural way of interacting with the system, where the user has to purposefully interact with the system. This will help to eliminate unintended commands issued by the user.

These constraints on image generation do however make the system more specialised, as only a subset of the total possible hand positions and rotations can reliably be identified. Given that the system is meant to be practically feasible, accuracy is valued higher than diversity.

Size of Training Data

As with all machine learning problems, more training data usually yields better classifiers. The ability to generate an infinite amount of data is thus advantageous. It does however increase the time and memory needed to train a classifier, limiting the size of the final training set.

In [7], 200 000 images are used to train a RDT for the final testing system. The RDTs of this project are trained using a maximum of 24 480 images per RDT, which included a variety of translations, rotations and scaling. The smaller training data set proved to be adequate for the constrained system, but a larger training set can further improve performance as indicated by the results in **Section 4.3**.

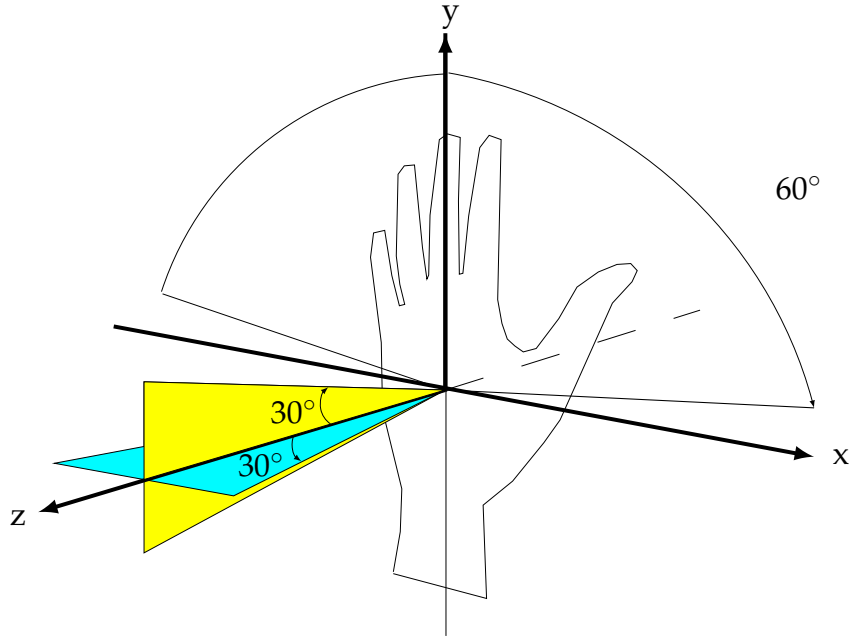


Figure 4.1.2: The pose constraints placed on the system, where the z-axis is perpendicular to the palm and the x- and y-axes are co-planar to the palm.

4.2 Trainer Design

This section discusses the design of the Random Decision Tree trainer. The first section discusses the depth features used to train the Region Classifier. **Section 4.2.3** and **Section 4.2.4** discusses the performance and memory issues that are relevant to the training process and how the trainer addresses these issues. **Section 4.2.5** discusses the data security concerns with the intermediate training results and how the trainer resolves them.

4.2.1 Depth Features

As described in the previous chapter, each internal node of a decision tree stores a split used to decide down which path a given data point is propagated. The region classifier built for this project uses oblique linear splits to propagate the data down the tree. The split type used is of the form:

$$F_{u,v}(I, \mathbf{x}) < \tau \quad (4.2.1)$$

$F_{u,v}(I, \mathbf{x})$ describes the depth feature at image coordinates \mathbf{x} of depth image I , while τ is the threshold value used to split the data based on the value of $F_{u,v}(I, \mathbf{x})$. The depth

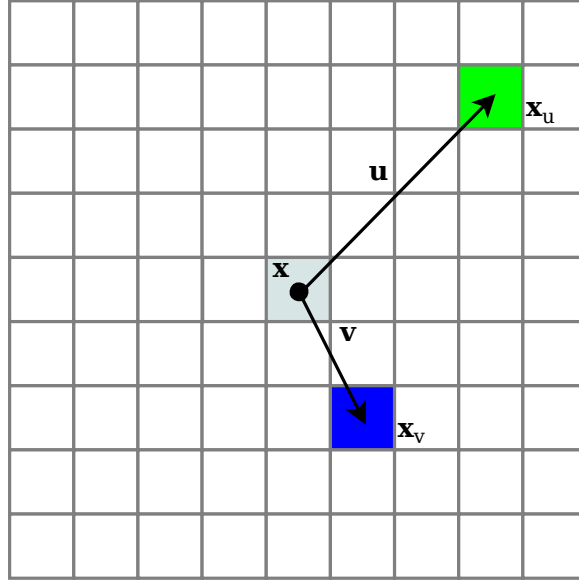


Figure 4.2.1: Illustration of the offset vectors \mathbf{u} and \mathbf{v} from the pixel located at \mathbf{x} . The offset pixels used for depth comparison is marked as \mathbf{x}_u and \mathbf{x}_v .

features are calculated as:

$$F_{\mathbf{u},\mathbf{v}}(I, \mathbf{x}) = I\left(\mathbf{x} + \frac{\mathbf{u}}{I(\mathbf{x})}\right) - I\left(\mathbf{x} + \frac{\mathbf{v}}{I(\mathbf{x})}\right) \quad (4.2.2)$$

The depth features compare the depth values of two pixels within a predefined neighbourhood of \mathbf{x} . The two pixels are specified using the offset vectors \mathbf{u} and \mathbf{v} , which can be used to define different variations of the depth feature $F_{\mathbf{u},\mathbf{v}}(I, \mathbf{x})$. The offset vectors are illustrated in **Figure 4.2.1**. The scaling factor $I^{-1}(\mathbf{x})$ makes the features invariant to depth translations. The combination of \mathbf{u} , \mathbf{v} and τ is unique for each internal node of a decision tree.

4.2.2 Random Split Generation

Chapter 3 describes how Random Decision Trees can be used to increase the effectiveness of combined classifiers, by using a randomisation algorithm to generate candidate splits. The increase in effectiveness and the results shown by [7] motivated the use of a random split generator to generate a set of 4000 unique splits when training a given node. For each random split the algorithm generates a pair of offset vectors and a threshold value. The lengths of the offset vectors are constrained to the integer range $[0, 60]$. The threshold value is randomly chosen from the range $[-0.2, 0.2]$, where 0.2 corresponds to a rough estimation of the average length of a hand in meters as proposed in [7].

The optimal split from the set of 4000 splits is defined as the split that produces the largest increase in information. The optimal split from the set of 4000 candidate splits is chosen by splitting the node training data using each split and scoring the two subsets using Equation (3.2.4) described in Section 3.2.3. The split which scores the lowest is chosen as the optimal split and is assigned to the current internal node.

The use of offset vectors with the applied scaling factor makes the depth feature invariant to 3D translations. The features are however not invariant to changes in rotation or the physical scale of the hand. This deficiency is overcome by using different rotation and scaling factors when generating the training data, as discussed in the next section.

4.2.3 Speed

The speed and efficiency of the trainer was an important consideration during the design of the trainer. Training of one RDT takes more than a week on a normal system, because of the large amount of data needed to train the RDTs and the depth of each RDT. The amount of memory needed by the trainer was also a concern if the classifier was to be trained on a normal system. These factors lead to the following design choices.

Programming Language

C++ was chosen as the program language for development because of its speed and open memory management system. The language made it possible to finely control the memory management of the application. This helped to avoid the unnecessary copying of large amounts of data, such as when a nodes training data is split and passed on to its children, which helped to increased speed and minimise memory usage.

Multithreading

A multithreaded system was implemented for training the classifier to allow the application to make full use of the processing power available to it. As was briefly discussed in Chapter 3, the data sets of two sibling nodes in the tree are independent from each other. This allowed for multiple nodes to be trained simultaneously, which lessened the training time for a single RDT.

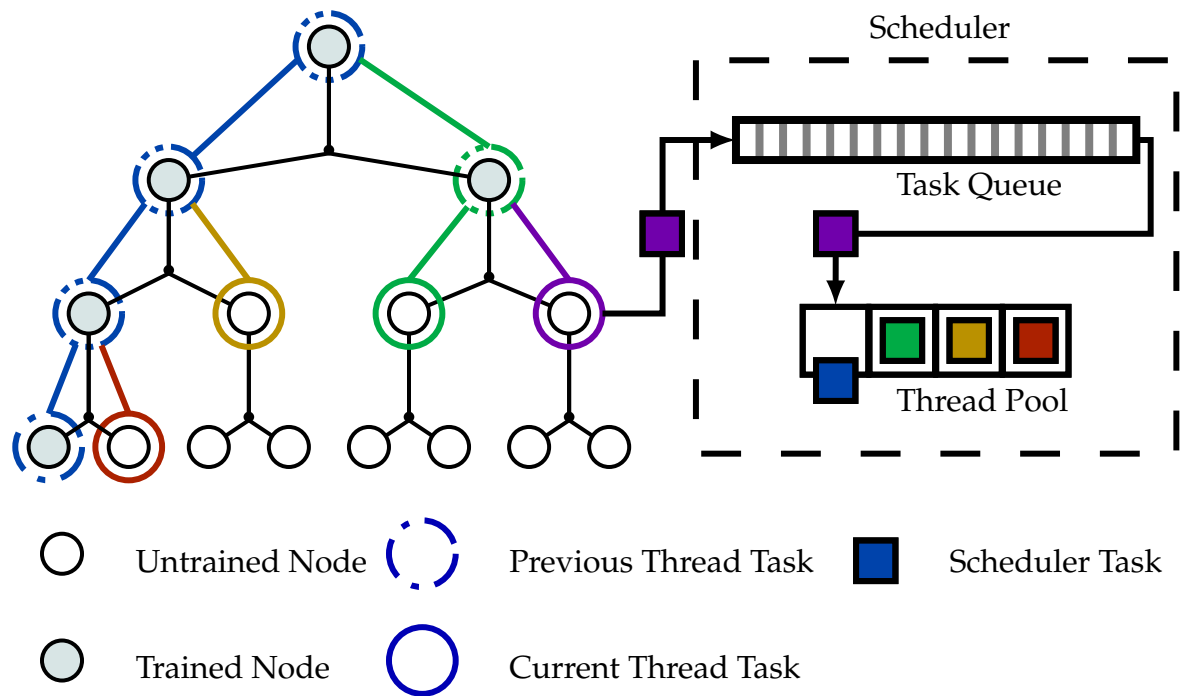


Figure 4.2.2: The scheduler used to manage threads.

A thread scheduler, shown in **Figure 4.2.2**, was implemented to manage the separate threads for training nodes. The scheduler consists of a queue of pending tasks and a pool of threads of a fixed sized. This ensures that the number of threads used for training is always constant and the overhead of creating new threads is avoided.

Once the training of a node finishes, the application will split the training data between its children. The training of the left child will continue in the same thread as that of its parent. The thread is released back to the scheduler once it reaches a leaf node. A task will be created to train the right child and will be added to the task queue in the scheduler. There it will wait until the scheduler assigns it to an open thread in the thread pool.

4.2.4 Memory Management

The large number of depth images needed to train a classifier makes it unpractical to keep all images loaded in memory during training. An image library was created to help facilitate the managing of images in memory. The functions of the library include loading an image from disk into memory, retrieving the image from memory for use by one or more training threads and releasing an unused image from memory when needed.

Hash Table

The image library was implemented using a hash table, because of the data structure's fast query times. The path to an image (represented as a string) serves as the key to the table. The path is hashed using a simple folding algorithm, where each set of four sequential 8-bit characters are concatenated into a 32-bit value. The 32-bit value for each set is then summed to form the hash key. Collisions are handled using a bucket system, where each bucket is represented as a linked list. If a collision is detected the original string is used to retrieve the correct entry inside the linked list.

Each entry in the table consists of a hash key, a collision string, a pointer to the corresponding image in memory and a reference counter. The reference counter is used to determine how many threads are currently operating on an image. This information is used to decide whether an image can be safely removed from the library.

The hash table is protected by a pair of mutexes which allows only one thread to operate on the table at a time. When a thread requests an image, it needs to retrieve a mutex lock. Once it has access to the lock, it will search the hash table for the existence of an image. If the image is not present in the library, the thread will load the image and store a corresponding entry in the library. If the image is already present, the thread will simply increment the reference count of the image.

Memory Allocation

Each image in the library is stored in a block of preallocated memory. The size of the memory is determined by the attributes and dimensions of the training images to be used. Our training system used 32-bit, one channel depth images with dimensions of 160x160 to train the classifier. When the library is initialised, a number of these memory blocks is allocated to the library as empty images. Each of the empty images are given a unique initial hash key and stored inside the hash table with a reference count of zero.

The library can help to reduce the frequency at which images are loaded into memory from disk. This is accomplished by storing images in memory even after the loading thread has finished its operations on the image. Since each image is used to train multiple nodes, there is a chance that another thread can make use of the cached image before it is loaded out of memory. The preallocated memory blocks also helps to lessen the time spent on allocating and deallocating memory for images.

4.2.5 Security

The number of images used to train a RDT meant that training of one tree took more than a week on a normal system. It was thus imperative to design the trainer to safely store its current progress at regular intervals, to avoid data loss caused by power outages or similar events. Furthermore, it should be able to quickly resume training from the last stored point.

Storage

The saving of the tree was simplified by storing the tree inside a vector. Each entry in the vector consists of the trained split for the node and the indices of the node's children within the vector. When a new node is created, it is pushed to the back of the vector and its index is stored in the parent node's entry. This structure eliminates the need for traditional pointers and makes serialising the tree easier for storage. Once a tree has been trained, it can easily be converted to the traditional pointer based tree structure.

Each time a trained node is added to the tree, a timer is consulted to check if the tree needs to be saved. Once the saving process is executed, any further processes which modifies the tree is halted. The tree is first saved to a temporary file. Once the saving process is completed, the old file is deleted and the temporary file is renamed. This ensures that there is always a copy of a previous valid save state if the application is interrupted during the saving process. The security of the data can be further improved by creating a copy of the tree on remote storage devices, but was not implemented for this project.

Fast Restore

It is possible to restore the previous training state of a tree using only the partially trained tree and splitting the original training data down the tree with the trained splits until leaf nodes are reached. This is however still a slow process, considering the larger training sets contain more than 40 000 images, with over a million data points spread across these images.

The restoration of the training state can be sped up significantly by also storing a data tree in parallel to the RDT. Each node of the data tree contains the training data of the corresponding node in the RDT. When training of a node finishes, the node's data is split into the left and right child data sets, which are then stored in the data tree at

Table of Image Generation Constraints		
	Constraint Set A	Constraint Set B (Extended Set)
Hand X-Rotation	$\pm 30^\circ$	$\pm 40^\circ$
Hand Y-Rotation	$\pm 30^\circ$	$\pm 40^\circ$
Hand Z-Rotation	$\pm 60^\circ$	$\pm 80^\circ$
Joint Angle X-Scale	± 0.06	± 0.07
Joint Angle Y-Scale	0.0	0.0
Joint Angle Z-Scale	± 0.06	± 0.07
Scale Hand	± 0.03	± 0.03
Scale Joints	± 0.085	± 0.095

Table 4.3.2: Constraints placed on the generated training and testing sets.

the appropriate indices. The data set of the trained node is emptied and released from memory in order to keep the size of the tree constant.

4.3 Testing

Various tests were conducted to determine the speed and accuracy of the region classifiers. The tests aimed to determine the optimal parameters for training a region classifier using the design discussed in this chapter. Furthermore, the tests were designed to also determine the accuracy of the classifiers when presented with data that fall outside of the designed set of constraints used to train the classifiers.

4.3.1 Experimental Setup

Two training sets, namely Training Set A and B, were generated using the image generator discussed in **Section 4.1**. The 17 poses are equally represented across the training sets. Different constraints are placed on each set, as shown in **Table 4.3.2**. The training portions of each set consists of 391680 unique images which are divided into 16 equal parts of 24480, where each part corresponds to one decision tree. Set A was further divided into four subsets of different sizes, while Set B was further divided into two subsets.

The number of images used to train one Random Decision Tree in each subset is shown in **Table 4.3.4**. A total of 16 RDTs were trained for each subset, which were used to construct 5 Random Decision Forests with sizes 1, 2, 4, 8, 16. A total of 30 classifiers were thus trained for testing purposes, as shown in **Table 4.3.3**. Note the unique label of each region classifier, which we shall use to identify specific region classifiers

Region Classifiers Trained and Tested			
Label	Number of Trees	Training Set (A or B)	Training Subset
RDF1_AT	1	A	Smallest
RDF1_AS	1	A	Small
RDF1_AM	1	A	Medium
RDF1_AL	1	A	Large
RDF2_AT	2	A	Smallest
RDF2_AS	2	A	Small
RDF2_AM	2	A	Medium
RDF2_AL	2	A	Large
RDF4_AT	4	A	Smallest
RDF4_AS	4	A	Small
RDF4_AM	4	A	Medium
RDF4_AL	4	A	Large
RDF8_AT	8	A	Smallest
RDF8_AS	8	A	Small
RDF8_AM	8	A	Medium
RDF8_AL	8	A	Large
RDF16_AT	16	A	Smallest
RDF16_AS	16	A	Small
RDF16_AM	16	A	Medium
RDF16_AL	16	A	Large
RDF1_BT	1	B	Smallest
RDF1_BL	1	B	Large
RDF2_BT	2	B	Smallest
RDF2_BL	2	B	Large
RDF4_BT	4	B	Smallest
RDF4_BL	4	B	Large
RDF8_BT	8	B	Smallest
RDF8_BL	8	B	Large
RDF16_BT	16	B	Smallest
RDF16_BL	16	B	Large

Table 4.3.3: The various region classifiers trained and tested.

in the rest of this document.

Two testing sets corresponding to the training sets A and B, namely Testing Set A and B, were generated using the same constraints shown in **Table 4.3.2**. Both sets contained 195840 images which are equally spread across the 17 poses.

4.3.2 Accuracy of Classifiers

The first set of tests evaluates the accuracy of different region classifiers, in order to review the effect of various training parameters on the final classification result and to

Number of RDT Training Images				
	Large (L)	Medium (M)	Small (S)	Smallest (T)
Set A	24 400	12 200	4 800	1 600
Set B	24 400	$n \setminus a$	$n \setminus a$	1 600

Table 4.3.4: Number of images per RDT in the subsets of Set A and Set B.

determine the optimal set of parameters which will give the most accurate classifier. The parameters evaluated include the size of the training set of a single Random Decision Tree and the number of individual classifiers used in a Random Decision Forrest. We also investigated the effect of changing the constraints placed on the training data and testing a classifier against a data set which includes data that fall outside of the training constraints.

Effect of training data size and forest size

We tested the 20 region classifiers trained using Training Set A on the corresponding Testing Set A. The size of the training data used to train individual RDTs and the number of RDTs used in the forests varies across the 20 classifiers. The results are shown in **Figure 4.3.1**. The annotations of these figures are rounded to the nearest percentage.

The top graph of **Figure 4.3.1** shows the average recognition rate across the 17 classes. It shows that increasing the number of training examples will increase the accuracy of the final classifier in general. Furthermore, the graph shows that increasing the size of a RDF will also increase the accuracy of the classifier, which confirms the results of [12] and [13]. RDF16_AL (16-tree classifier trained using the Large subset of Training Set A) is the best performing region classifier with a recognition rate of 74.53%.

The larger hand regions are better represented in the training and test sets than others, since they have a larger pixel count. Another measurement was taken to take into account these differences in pixel representation of the different regions. The bottom graph of **Figure 4.3.1** shows the results of the top graph weighted using the pixel count of each region and then averaged across the classes. The results still show the same general trend, where increasing the size of the training set and the forest classifier will generally increase the accuracy of the final classifier. The best performing classifier had a weighted recognition rate of 81.69%.

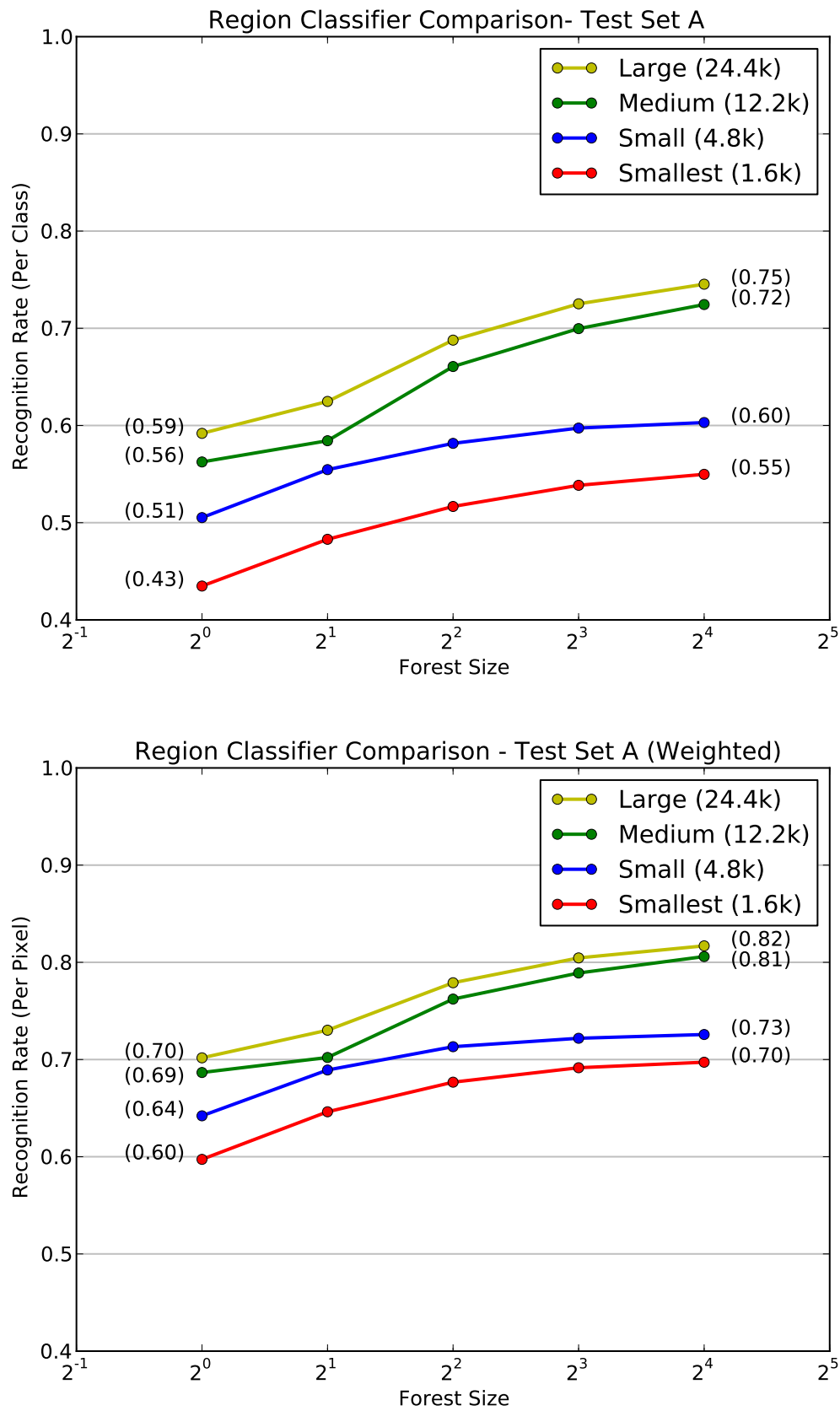


Figure 4.3.1: Results of the Large and Smallest Region classifiers for both training sets A and B on testing set B. The first graph shows the per class results while the second graphs shows the same results that are weighted based on the number of pixels present for each class.

The non-weighted results give a better representation of how well the classifiers can identify individual hand regions compared to the weighted results, while the weighted results give a better representation of the overall performance of a classifier. The top confusion matrix of **Figure 4.3.6** shows the test results of testing the 16-tree classifiers from Training Set A on Testing Set A in matrix form, which is discussed in depth later on in this section.

Effect of Constraints

The next set of tests were designed to determine the effect of placing constraints on the generation of the synthetic training data. Two sets of constraints were used to generate training and testing data, as previously discussed in **Section 4.3.1** and shown in **Table 4.3.2**. Set A is the initial set of constraints that was used to design the region trainer. Set B is an extension of A, which allows for a larger range of possible views and poses of the hand. Set A is thus more specialised as opposed to Set B which is more general.

The first set of constraint tests were performed on testing data generated using constraint Set A. For these test we used region classifiers trained using the Large and Smallest subsets of both Training Sets A and B. The results of these tests are shown in **Figure 4.3.2**. The top graph shows the non-weighted results while the bottom graph shows the weighted results. As expected, the specialised classifiers trained using constraint Set A generally performs better than those trained using Set B, with RDF16_AL achieving a recognition rate of 74.54% and 81.69% for the non-weighted and weighted results respectively. An unexpected result comes from the classifiers trained using the Smallest subset. The 8 and 16 tree classifiers of Training Set B perform the same or better than the specialised classifiers from Training Set A when using the Smallest subset. This is probably due to the small amount of training images used and requires further investigation.

The second set of constraint tests (shown in **Figure 4.3.3**) were performed on the extended testing data generated using constraint Set B, using a similar setup as described in the previous paragraph. As expected, there is a noticeable drop in accuracy for all the classifiers compared to the previous set of tests. Furthermore, the more generalised classifiers trained using Training Set B performed more accurately than the corresponding classifiers trained using Training Set A. The best performing classifier for this test set is RDF16_BL with recognition rates of 65.39% and 74.12% for the non-weighted and weighted results respectively.

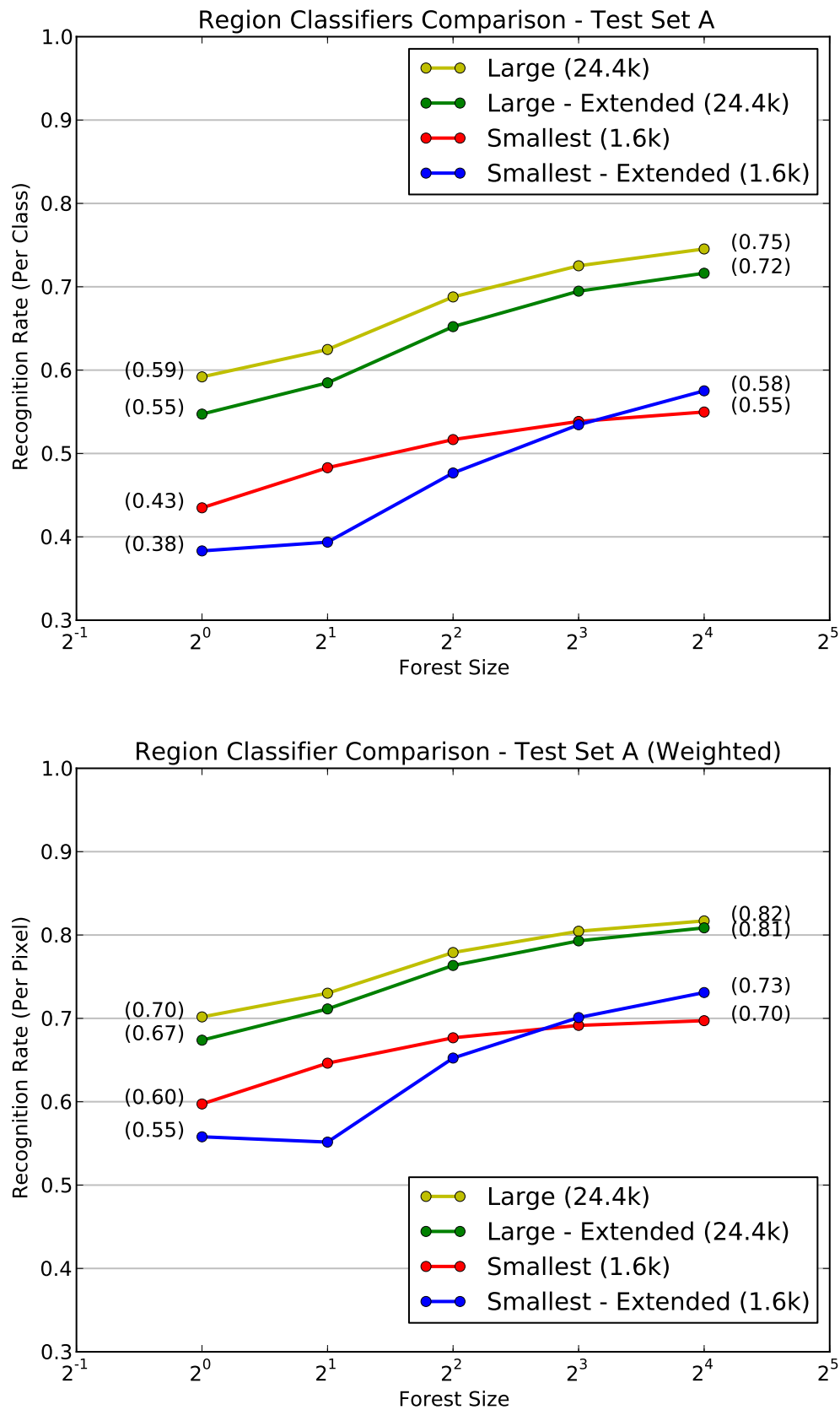


Figure 4.3.2: Results of the Large and Smallest Region classifiers for both training sets A and B on testing set A. The first graph shows the per class results while the second graphs shows the same results that are weighted based on the number of pixels present for each class.

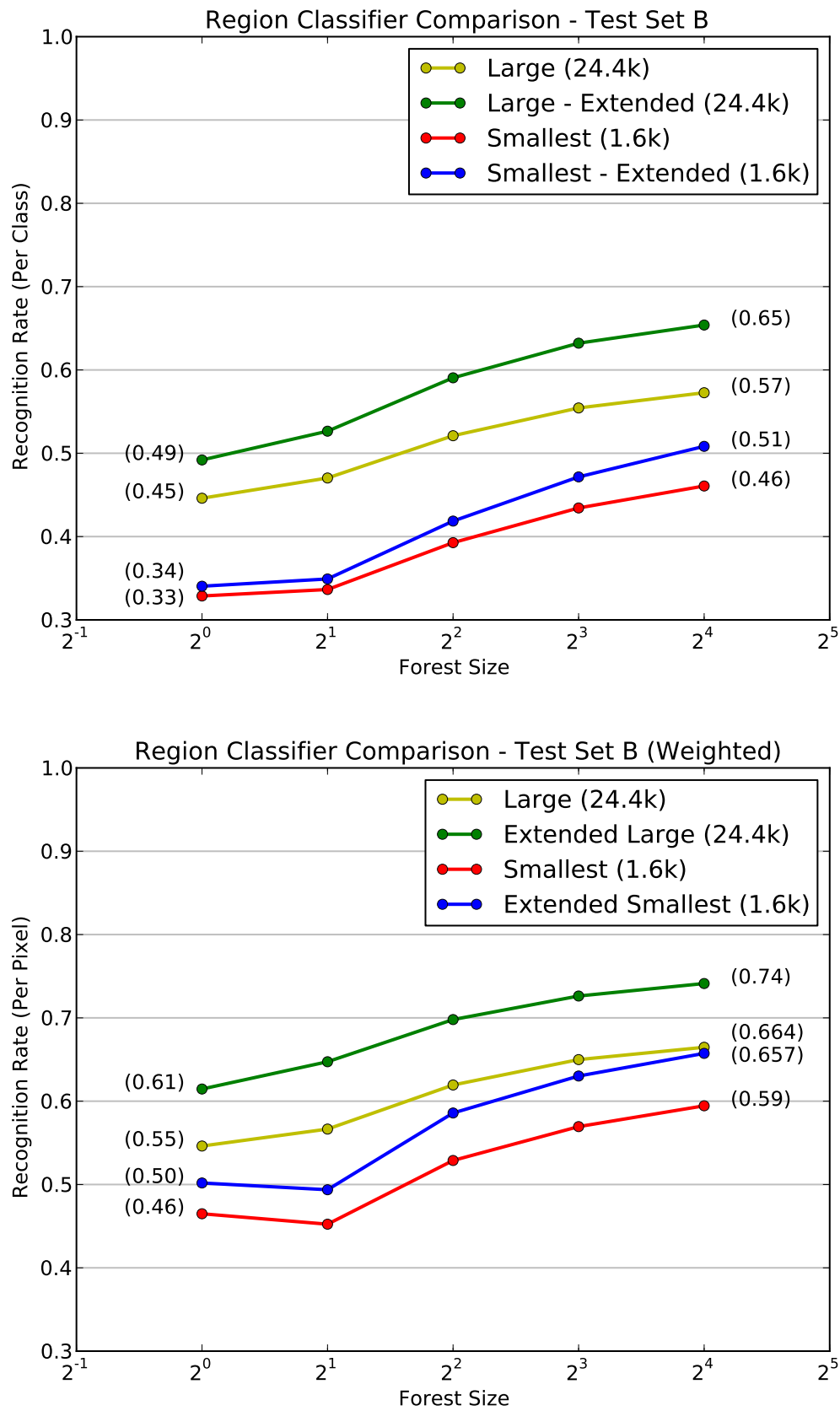


Figure 4.3.3: Results of the Large and Smallest Region classifiers for both training sets A and B on testing set B. The first graph shows the per class results while the second graphs shows the same results that are weighted based on the number of pixels present for each class.

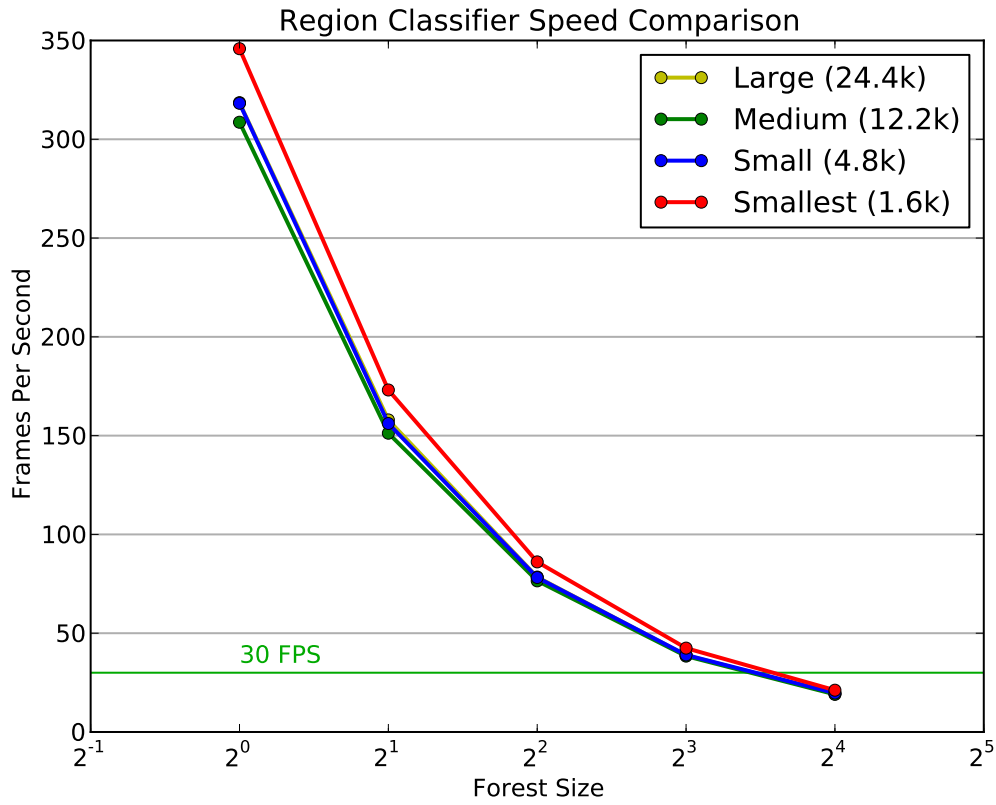


Figure 4.3.4: Speed test of the various classifiers trained using training set A.

4.3.3 Speed of Classifiers

The final set of tests evaluated the average classification speed of various region classifiers. The 20 classifiers trained using Training Set A was used to classify the 195840 depth images of Testing Set A. The speed of the classifiers was measured in Frames Per Second (FPS), calculated using:

$$FPS = \frac{\text{Number of Depth Images Classified}}{\text{Classification Time in Seconds}}$$

Figure 4.3.4 shows the results of the speed tests. It can be seen that the classifiers trained using the Smallest training subset performed the fastest. This can be attributed to the relatively low complexity of these classifiers compared against the other classifiers. The similar performance of the other classifiers indicate that the size of the training data does not in general affect the speed of the final classifier.

The number of trees does however have a considerable effect on the speed of the classifier. All classifiers were able to perform in real-time (30+ FPS), with the exception of the 16-tree classifiers. It is thus preferable for our system to use classifiers with 8 trees or less.

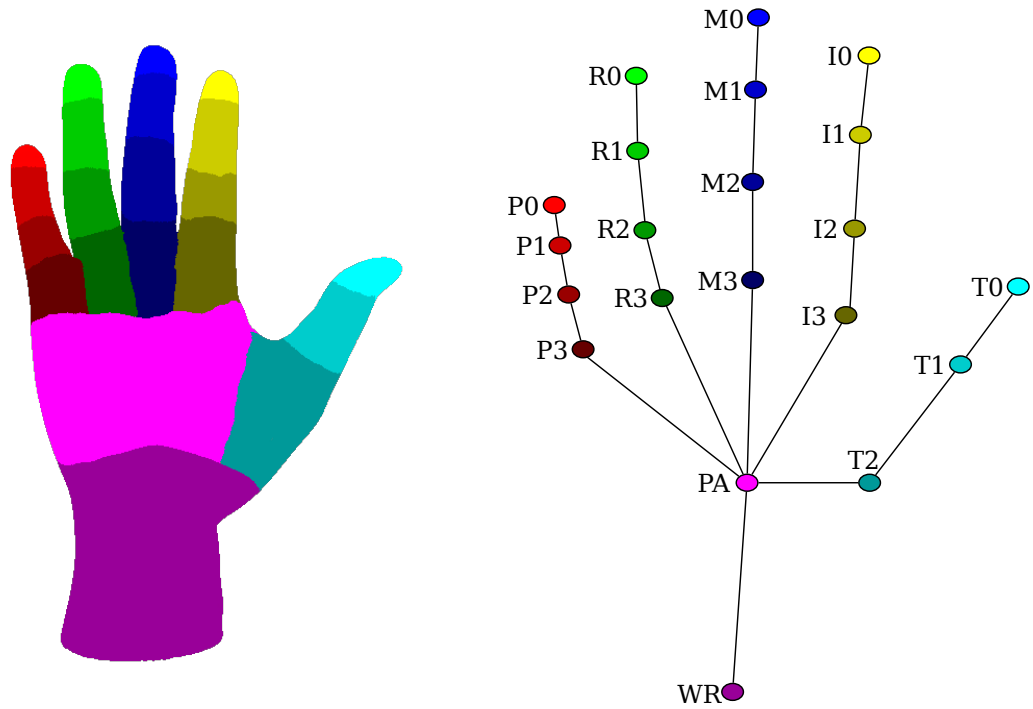


Figure 4.3.5: The labels of the various joints corresponding to the 21 regions.

4.3.4 Confusion Properties

Confusion matrices were constructed using the results from the constraint tests for the RDF16_AL and RDF16_BL region classifiers. Colour graphs of these matrices are shown in **Figure 4.3.6** and **Figure 4.3.7**. The axes of each graph are labelled using the joint labels shown in **Figure 4.3.5**, where each joint corresponds to one of the 21 hand regions. The values of the colours are indicated on the colour bar to the right of the graphs. The values correspond to the recognition rate and are calculated as:

$$value = \frac{\text{number of predicted samples}}{\text{number of actual samples}}$$

The graphs of **Figure 4.3.6** show the confusion matrices when testing region classifier RDF16_AL on Testing Sets A (top graph) and Testing Set B (bottom graph). The diagonal of the top graph shows that the classifier performs well when tested against Testing Set A, struggling the most with regions that represent the little finger and the tip of the other fingers. This is expected, since these regions are typically small and under represented in the training sets. The inverse is in fact true, since the large palm, wrist and thumb regions were classified the most accurately. The results of testing RDF16_AL on Testing Set B are less accurate, as shown by the mostly yellow diagonal.

One of the notable features found in the confusion graphs are the two lines that run in parallel with the diagonal with an offset of three cells. A cursory inspection will show that these lines represent the region samples which the region classifier con-

fuses with adjacent regions found on an adjacent finger. This typically occurs when the region classifier is presented with one of the “closed” poses, where the fingers are generally in close contact to each other. These parallel lines are especially distinct on the bottom graph, where the classifier does not cope well enough with the less restrictive constraints of Testing Set B.

Another notable feature found on the graphs is the vertical pattern of cells seen in the column labelled PA. These cells indicate that the finger tips (labels *0) and knuckle regions (labels *3) are many times confused with the palm region. There are two reasons why the confusions occur. Firstly, the palm region is adjacent to the knuckle regions, where the pixels near the region borders are generally hard to classify correctly. Secondly, the fingers tips come into contact with the palm in the majority of tested poses, where the pixels are again harder to classify correctly. Again, this feature is most prevalent in the bottom graph.

The last notable feature that can only be seen in the bottom graph of **Figure 4.3.6** is a slight vertical line in the column labelled WR. This line shows that some of the regions are confused with the wrist region when the classifier is tested against a less constrained testing set. These confusions mostly occur because of the increased range of viewing angles of the hand. The region classifiers trained using Testing Set A associates the lower part of the image with the wrist region, which is not always true with the less constrained Testing Set B, where a wider range of rotations can be applied to the hand.

The confusion graphs of **Figure 4.3.7** show the results of testing region classifier RDF16_BL on Testing Sets A (top graph) and B (bottom graph). These graphs contain the same notable features described in the previous paragraphs. These features are however less prominent on the bottom graph, indicating that the classifier can cope better when tested against a less constrained testing set, while still performing similar to RDF16_AL.

In summary, the results showed in **Figure 4.3.6** and **Figure 4.3.7** indicate that the tested region classifiers confused adjacent regions most, while also struggling to correctly classify smaller hand regions. Furthermore, the classifiers do not perform accurately when presented data which falls outside the constraints of the training data.

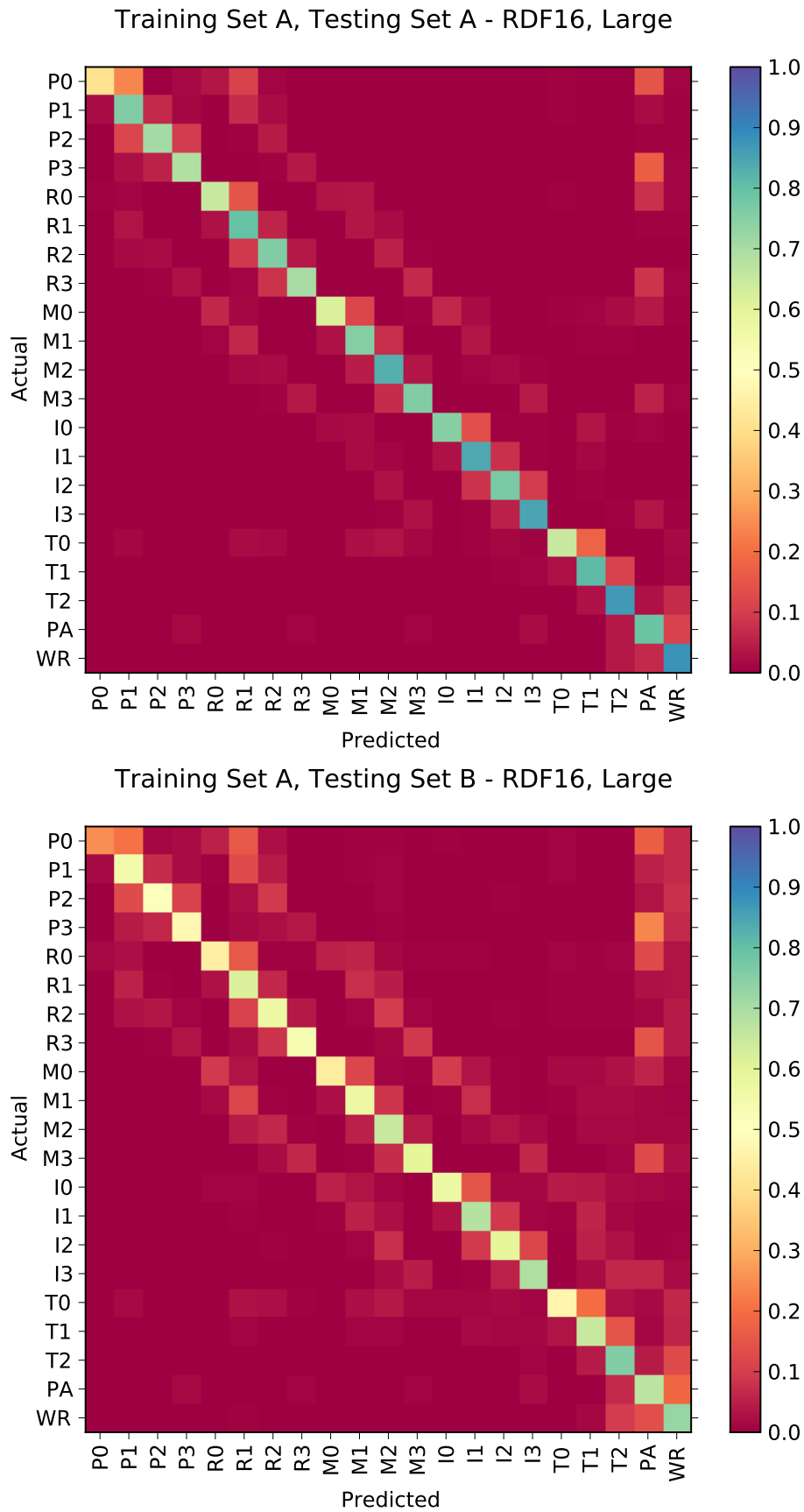


Figure 4.3.6: Confusion matrices of RDF consisting of 16 trees trained using the Large data set of Training Set A. The top confusion matrix shows the results of testing the classifier on Testing Set A, while the bottom matrix shows the results while testing with Test Set B.

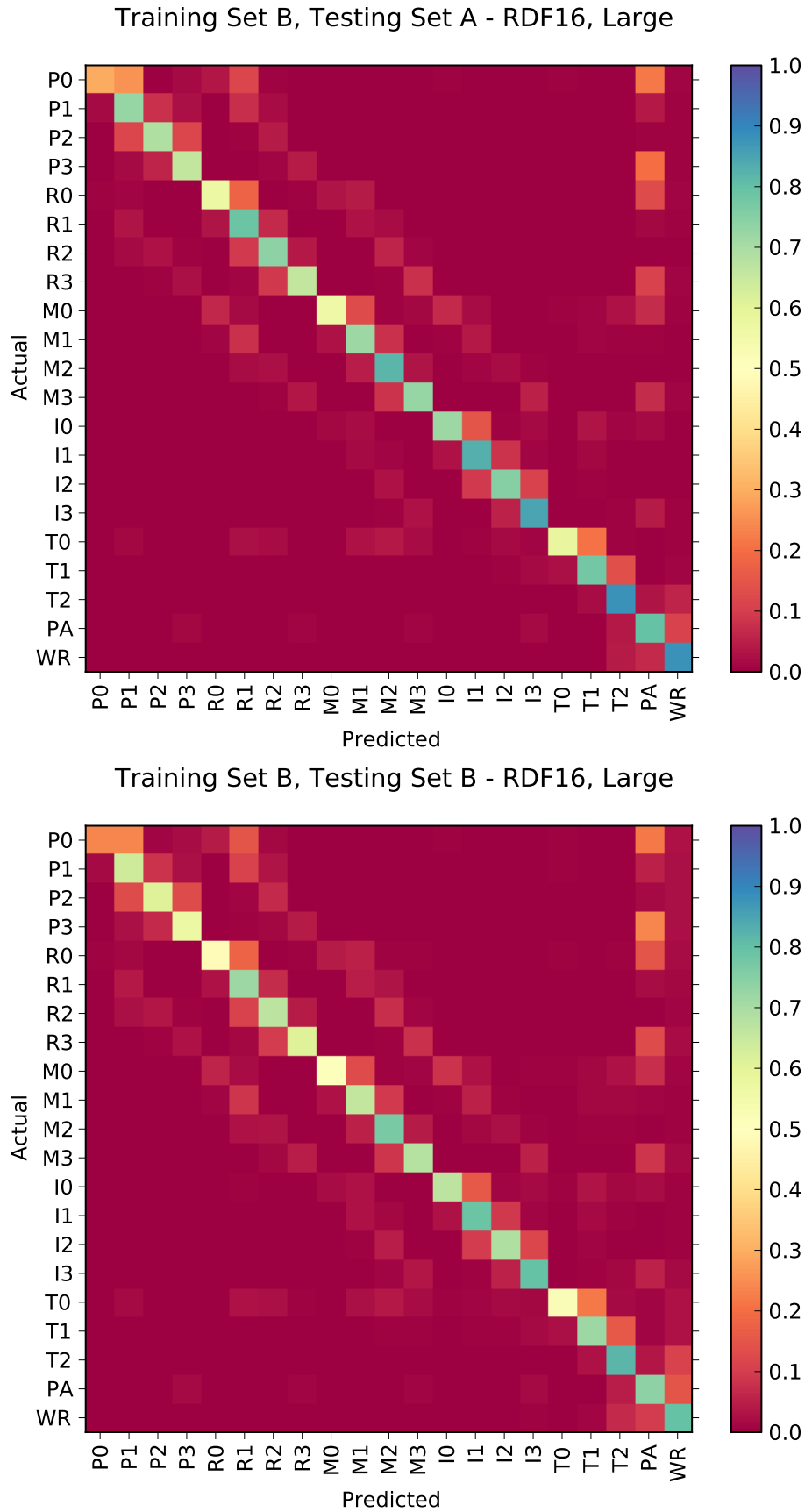


Figure 4.3.7: Confusion matrices of RDF consisting of 16 trees trained using the Large data set of Training Set B. The top confusion matrix shows the results of testing the classifier on Testing Set A, while the bottom matrix shows the results while testing with Test Set B.

4.4 Summary

This chapter described the implementation of the Region Classifier for this system. The generation of the synthetic images for training and testing was described in **Section 4.1**. **Section 4.2** described the design of Random Decision Tree trainer, which included the depth features used and various performance and data security issues that are addressed by the trainer.

Section 4.3 showed the results of the various tests performed to ensure that the Region Classifier worked as expected. The results indicated that using larger data sets for training increases the accuracy of the resulting classifier. Also, using constrained training data sets delivers a more specialised classifier that performs well on similarly constrained testing data, but performs poor for less constrained data. Furthermore, combining individual trees into a single forest classifier results in a more general classifier that performs better on testing data, but also increases the complexity and classification time of the classification process.

Chapter 5

Joint Position Estimation

Once the per pixel classifier has classified each of the depth image pixels, the location of the 3D joints can be estimated. If the accuracy of the pixel classifier is high enough, a simple centre of gravity algorithm can be used to find the 2D joint locations. However, as was shown in the previous chapter, the final result from the region classifier contains a large amount of outliers, making a centre of gravity approach inaccurate. An alternative approach proposed by [7] and [2] is to use the Mean Shift Algorithm for joint position estimation.

5.1 Centre of Gravity Joint Estimation

A joint estimator that uses a centre of gravity (COG) algorithm to find the joint locations was implemented for testing purposes. The algorithm is simple to implement, computationally efficient and provides relatively good estimations if the classification from the Region Classifier is accurate.

The pixel probabilities retrieved from the Region Classifier are used to assign a class corresponding to one of the 21 joints to each pixel in the depth image. A simple maximum probability classification is used to assign classes to pixels, where the class with the highest probability for each pixel is assigned to that pixel. The 2D location of the joint j belonging to class i can then be found using the following formulae

$$j_{c,x} = \frac{1}{N_c} \sum_{i=1}^{N_c} x_{c,i} \quad (5.1.1)$$

and

$$j_{c,y} = \frac{1}{N_c} \sum_{i=1}^{N_c} y_{c,i} \quad (5.1.2)$$

where N_i is the number of pixels belonging to class c , with x_{c_i} and y_{c_i} being the x- and y-components of pixel i belonging to class c .

The above algorithm uses hard decisions to decide the location of the joints and can perform poorly when there are many pixel outliers, and was implemented only to use as a baseline for testing. Shotton [2] et al. and Keskin [7] et al. proposed using the Mean-Shift algorithm instead for joint estimation, as it is less affected by outliers. The Mean-Shift algorithm is described in the next section.

5.2 Mean-Shift Algorithm

The Mean-Shift algorithm was first described by Fukunaga and Hostetler [44]. Their paper described a technique to find the modes of an unknown probability density function. The technique developed is extremely fast, as it does not involve the calculation of the density function or the gradient of the density function. Comaniciu and Meer [45] revisited the work done by [44] and showed how the Mean-Shift algorithm can be used to solve various Computer Vision tasks efficiently.

Fukunaga and Hostetler [44] make use of the multivariate kernel density estimators of [46] to estimate the gradient of an unknown density function. Using a special class of kernels, they develop a formula for estimating the gradient of the density function, which contains the Mean-Shift. The formula shows that the modes of the density function can easily be found using the Mean-Shift.

5.2.1 Density Estimator

This section will quickly summarise how the formula for the Mean-Shift is derived from the kernel density estimators of [46] as shown by [45], since their approach is more relevant to our system. Given a set of d -dimensional data points $[\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$ in the space R^d , the probability density function at the point \mathbf{x} can be estimated as:

$$\hat{p}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \quad (5.2.1)$$

The parameter h is chosen as the bandwidth of the kernel, which will control the number of samples n seen by the kernel. The multivariate function $K(\mathbf{x})$ is the kernel

function which satisfies the following conditions:

$$\int_{R^d} K(\mathbf{x}) d\mathbf{x} = 1 \quad (5.2.2)$$

$$\lim_{\|\mathbf{x}\| \rightarrow \infty} \|\mathbf{x}\|^d K(\mathbf{x}) = 0 \quad (5.2.3)$$

$$\int_{R^d} \mathbf{x} K(\mathbf{x}) d\mathbf{x} = 0 \quad (5.2.4)$$

$$\int_{R^d} \mathbf{x} \mathbf{x}^T K(\mathbf{x}) d\mathbf{x} = c_K \mathbf{I} \quad (5.2.5)$$

where c_K is a constant and \mathbf{I} is the identity matrix. The above conditions can be satisfied using the radially symmetric kernel function

$$K(\mathbf{x}) = c_{k,d} k(\|\mathbf{x}\|^2) \quad (5.2.6)$$

where $c_{k,d}$ ensures K integrates to 1, and the scalar function $k(\mathbf{x})$ is known as the kernel profile. Substituting (5.2.6) into (5.2.1) produces the final density estimator:

$$\hat{p}_K(\mathbf{x}) = \frac{c_{k,d}}{nh^d} \sum_{i=1}^n k\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right) \quad (5.2.7)$$

5.2.2 Density Gradient Estimator

Fukunaga estimates the gradient of the density function by finding the gradient to the estimated density function. The gradient estimator derived from (5.2.7) is

$$\begin{aligned} \hat{\nabla}_{\mathbf{x}} p(\mathbf{x}) &\equiv \nabla_{\mathbf{x}} \hat{p}_K(\mathbf{x}) \\ &\equiv \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^n (\mathbf{x} - \mathbf{x}_i) k' \left(\left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2 \right) \end{aligned} \quad (5.2.8)$$

Fukanaga then proceeds to derive various conditions on $K(\mathbf{x})$ to ensure that the gradient estimator is asymptotically unbiased, consistent in a mean-square sense and uniformly consistent [44].

5.2.3 Introduction of the Mean-Shift

Assuming the derivative of $k(\mathbf{x})$ exists, we can define a function

$$g(\mathbf{x}) = -k'(\mathbf{x}) \quad (5.2.9)$$

The kernel function can be rewritten as

$$G(\mathbf{x}) = c_{g,d} g(\|\mathbf{x}\|^2) \quad (5.2.10)$$

where $c_{g,d}$ is a normalisation constant. We shall denote the radial distance of a sample i from the kernel centre as

$$\sigma_i = \left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2 \quad (5.2.11)$$

Substituting (5.2.9) and (5.2.11) into (5.2.8) gives

$$\begin{aligned} \nabla_{\mathbf{x}} \hat{p}_K(\mathbf{x}) &= \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^n (\mathbf{x}_i - \mathbf{x}) g(\sigma_i) \\ &= \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^n [g(\sigma_i) \mathbf{x}_i - g(\sigma_i) \mathbf{x}] \\ &= \frac{2c_{k,d}}{nh^{d+2}} \left[\frac{\sum_{i=1}^n g(\sigma_i)}{\sum_{i=1}^n g(\sigma_i)} \right] \left[\sum_{i=1}^n g(\sigma_i) \mathbf{x}_i - \sum_{i=1}^n g(\sigma_i) \mathbf{x} \right] \\ &= \frac{2c_{k,d}}{nh^{d+2}} \left[\sum_{i=1}^n g(\sigma_i) \right] \left[\frac{\sum_{i=1}^n g(\sigma_i) \mathbf{x}_i}{\sum_{i=1}^n g(\sigma_i)} - \frac{\sum_{i=1}^n g(\sigma_i)}{\sum_{i=1}^n g(\sigma_i)} \mathbf{x} \right] \\ &= \frac{2c_{k,d}}{h^2} \left[\frac{1}{nh^d} \sum_{i=1}^n g(\sigma_i) \right] \left[\frac{\sum_{i=1}^n g(\sigma_i) \mathbf{x}_i}{\sum_{i=1}^n g(\sigma_i)} - \mathbf{x} \right] \end{aligned} \quad (5.2.12)$$

The first term of the product in (5.2.12) is simply (5.2.7) using the kernel function $G(\mathbf{x})$ with an adjusted normalisation factor

$$\hat{p}_G(\mathbf{x}) = \frac{c_{g,d}}{nh^d} \sum_{i=1}^n g\left(\left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2\right) \quad (5.2.13)$$

while the second term is the Mean-Shift vector

$$\mathbf{m}_G(\mathbf{x}) = \frac{\sum_{i=1}^n g\left(\left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2\right) \mathbf{x}_i}{\sum_{i=1}^n g\left(\left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2\right)} - \mathbf{x} \quad (5.2.14)$$

It is shown by [44] and in a more general form by [45] that the Mean-Shift vec-

tor $\mathbf{m}_G(\mathbf{x})$ always points in the direction of maximum ascent. It is also shown that continuously shifting the kernel using the Mean-Shift will result in convergence at a local mode of the density function. The Mean-Shift can thus be used to quickly find the modes of an unknown density function, without actually estimating the density or gradient functions.

5.3 System Implementation

Following the examples from [7] and [2], the Mean-Shift algorithm was used to estimate the 2D image coordinates of the hand joints. This section will describe the algorithm implemented in our system.

5.3.1 Calculating the Mean-Shift

The per-pixel region classifier will assign a set of soft class probabilities to each pixel in the image. These probabilities can be used to generate a weight image for each class. Keskin proposes multiplying the probability of each pixel with the square of the depth at that pixel, since pixels further from the camera represent a larger physical area. The resulting image is then further smoothed using a Gaussian kernel [7].

The weight image for each class is calculated as:

$$\omega_c(\mathbf{x}) = \begin{cases} P(c|I, \mathbf{x}) \cdot I^2(\mathbf{x}), & \text{foreground} \\ 0, & \text{background} \end{cases} \quad (5.3.1)$$

where $I(\mathbf{x})$ is the value at pixel \mathbf{x} of depth image I . The weight image is zero at each pixel which represents the background in the depth image I . Each sample of the weight image is smoothed using the Gaussian kernel:

$$g(\sigma_i) = 1/2\pi \exp(-\sigma_i/2) \quad (5.3.2)$$

where σ_i is the sample radial distance as given by (5.2.11). Using the above we can calculate for each class c the Mean-Shift vector from pixel \mathbf{x} as:

$$\mathbf{m}_c(\mathbf{x}) = \frac{\sum_{i=0}^n \omega_c(\mathbf{x}_i) g(\sigma_i) \mathbf{x}_i}{\sum_{i=0}^n \omega_c(\mathbf{x}_i) g(\sigma_i)} - \mathbf{x} \quad (5.3.3)$$

5.3.2 Estimating the Hand Joints

Once the weight image ω_c has been calculated for class c , (5.3.3) can be used to find the modes of ω_c , which represent the potential points at which the hand joint associated with the class is located. One method of finding the nodes involves uniformly placing

a number of kernels with fixed bandwidths across the image and shifting the kernels using \mathbf{m}_c until they converge. The area at which the most kernels converge is then chosen as the 2D location of the hand joint.

The technique used for this system uses only one kernel and tries to shift it to the most dense region in the weight image. The kernel is initially placed in the centre of the image and given a bandwidth of half the image width. The Mean-Shift vector is calculated and used to shift the kernel towards the perceived area of greatest density. The bandwidth of the kernel is then halved and the Mean-Shift vector is recalculated. This process is continued until the kernel either does not move between two iterations or the bandwidth reaches a value of 1.

This approach works well for most of the classified images, since there is usually only one prominent area in each weight image. The initial iteration considers the image as a whole, while later iterations focus more on the immediate area surrounding the kernel at its current location. This allows the algorithm to find the hand joint faster and more efficiently, while still remaining moderately accurate.

The technique does however become inaccurate when the weight image contains more than one prominent area. Such weight images are usually the result of poor classification of classes belonging to small or occluded hand regions in the classified depth image. The class probabilities of these regions, while small, are usually spread across the hand, making the Mean-Shift algorithm inaccurate. The next section proposes a possible solution to this problem by modifying the weight image.

5.3.3 Joint Reservation Algorithm

A modification of the algorithm described above was developed which attempts to partially solve the problems associated with occluded joints and weak per pixel classification of the hand regions. The modified algorithm is based on heuristics unique to our problem and only attempts to solve a few issues specific to the joint estimation process of this project.

The algorithm calculates a confidence measurement for all joint estimations as calculated by the Mean-Shift algorithm. The most confident joint is kept as is and the rest is re-estimated. In this way it "places" joints sequentially on the depth image from most to least confident. It attempts to place joints in areas not occupied by previously placed joints, while still trying to place adjacent joints in close proximity to each other.



Figure 5.3.1: The green rectangle represents the smallest window needed to enclose all the foreground pixels in the given label image.

This section will show how the confidence of an estimated joint is calculated. It will also show how the weight image ω_c is modified to deal with occlusions and an inaccurate classification from the per pixel region classifier.

Confidence Calculation

We define the confidence of a joint belonging to class c at the 2D image coordinates \mathbf{x} as the average of the probabilities in the neighbourhood of the joint:

$$f_c(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n P(c|I, \mathbf{x}_i) \quad (5.3.4)$$

where \mathbf{x}_i for all values of i lie within the window centred at \mathbf{x} . The size of the window is different for each class, since some classes represent a smaller region of the hand. The window size h_c is calculated as

$$h_c = \begin{cases} \lceil k_w P(c) \rceil, & \lceil k_w P(c) \rceil \geq k_{min} \\ k_{min}, & \lceil k_w P(c) \rceil < k_{min} \end{cases} \quad (5.3.5)$$

where

$$k_{min} = c_{min} k_w$$

$P(c)$ is the probability of the class as found in the training data used to train the per pixel classifier. The integer k_w is the larger of the two dimensions (width/height) of the smallest possible rectangular window needed to enclose all the foreground points in the depth image, as shown in **Figure 5.3.1**. This gives a small measure of scale invariance to all of the calculations which uses h_c . The constant c_{min} was chosen as 0.15 for the system developed and ensures that some of the smaller classes are always given a window size that still produces an effect.

Instead of using the class probabilities as is, we use the smoothed weight image as described in **Section 5.3.1**:

$$f_{c,\omega}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \omega_c(\mathbf{x}_i) g(\sigma_i) \quad (5.3.6)$$

Equation (5.3.6) ensures that probabilities closer to the joint with respect to the weight image contributes more to the confidence calculation. We also take the physical area of each pixel into account by using the weight image.

The hand regions are defined in such a way that the joints are located in the centre of the region. Furthermore, the surface of regions can be approximated using planes. Joints can thus not be located near discontinuities, such as the edge of the palm or finger. We also expect each pixel belonging to the same class as the joint to have a similar depth to that of the joint, given the small distance between the joints and their corresponding region pixels. The above constraints can be taken into account by modifying Equation (5.3.6) to also include the difference in depth between the joint and the surrounding pixels:

$$f_{c,\delta}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \frac{\omega_c(\mathbf{x}_i) g(\sigma_i)}{\delta(\mathbf{x}, \mathbf{x}_i)} \quad (5.3.7)$$

where

$$\delta(\mathbf{x}, \mathbf{x}_i) = 1 + |I(\mathbf{x}) - I(\mathbf{x}_i)|^2$$

Placed Joint Weight Image Modification

Once the confidence of each joint has been calculated, the joint with the highest confidence can be placed. The area around the placed joint is then reserved for that joint, since any two joints of a hand are always separated by a set distance. The size of the reserved area is different for each class of joint, since some joints are further spaced apart than others (palm joint vs. finger tip).

An image is used to keep track of which parts of the hand have previously been reserved by joints. It can be used to adjust the weight image of the unplaced joints to ensure that they are less likely to be placed within a reserved region. Such a reservation image is filled with values of 1 at creation. When a hand joint is placed, an impression of the form of a Gaussian distribution is made in the reservation image centred at the 2D image coordinates of the joint. The effect of the impression is calculated as

$$\epsilon_c(r) = 1 - 0.5e^{-\left(\frac{2r}{h_c}\right)^2} \quad (5.3.8)$$

where r is the Euclidean distance between the 2D image coordinates of the joint belonging to class c and the neighbouring pixel which is affected by the impression. The window size h_c is used to ensure that the impression is representative of the size of the hand joint belonging to class c . The reservation image is calculated as

$$E(\mathbf{x}) = E_1(\mathbf{x}) \prod_{c \in S_e}^{N_e} \epsilon_c(\|\mathbf{x} - \mathbf{x}_c\|) \quad (5.3.9)$$

where E_1 is the original image filled with ones and \mathbf{x}_c is the 2D image coordinates of the joint belonging to class c , which itself is an element of the set of all placed joints S_e . N_e represents the number of joints which have already been placed. $E(\mathbf{x})$ is recalculated each time a new joint is placed, before re-estimating the positions of the unplaced joints.

Adjacent Joint Weight Image Modification

As previously discussed, occlusion of the hand joints poses a problem when estimating the position of the hand joints. The result of the per pixel region classifier will contain little, if any, information on the location of an occluded joint, while the information present will in most cases be inaccurate. If only a small part of the hand is occluded, we can find the general location of the occluded joints by using the location of any joints adjacent to the occluded joints that have already been placed.

An occluded joint is more likely to be found in the neighbourhood surrounding adjacent joints. The effect of an adjacent joint is calculated as

$$a_{c,i}(r) = 1 + e^{-\left(\frac{r}{h_i}\right)^2} \quad (5.3.10)$$

where i is an element of the set S_{ac} , which consists of the placed joints which are adjacent to the joint of class c . The total effect of all the joints in S_{ac} on c is calculated as

$$A_c(\mathbf{x}) = \prod_{i \in S_{ac}}^{N_{ac}} a_{c,i}(\|\mathbf{x} - \mathbf{x}_i\|) \quad (5.3.11)$$

Theoretical Effect of the Weight Image Modification

The modified weight image of class c can be written as

$$\omega_{c,M}(\mathbf{x}) = \omega_c(\mathbf{x})M_c(\mathbf{x}) \quad (5.3.12)$$

where $M_c(\mathbf{x})$ is a function consisting of the reservation image $E(\mathbf{x})$ and the adjacent

joint image $A_c(\mathbf{x})$:

$$\begin{aligned} M_c(\mathbf{x}) &= E(\mathbf{x})A_c(\mathbf{x}) \\ &= E_1(\mathbf{x}) \prod_{c \in S_e}^{N_e} \epsilon_c(\|\mathbf{x} - \mathbf{x}_c\|) \prod_{i \in S_{ac}}^{N_{ac}} a_{c,i}(\|\mathbf{x} - \mathbf{x}_i\|) \end{aligned} \quad (5.3.13)$$

As can be seen from (5.3.13), the modification will have no effect on the first joint placed. This is as expected, since the first joint is assumed to be the most accurate. This joint will serve as the anchor joint, which will affect the placement of the rest of the joints. As each successive joint is placed the areas in which the remaining joints can be placed is reduced, making placement of these uncertain joints potentially more accurate.

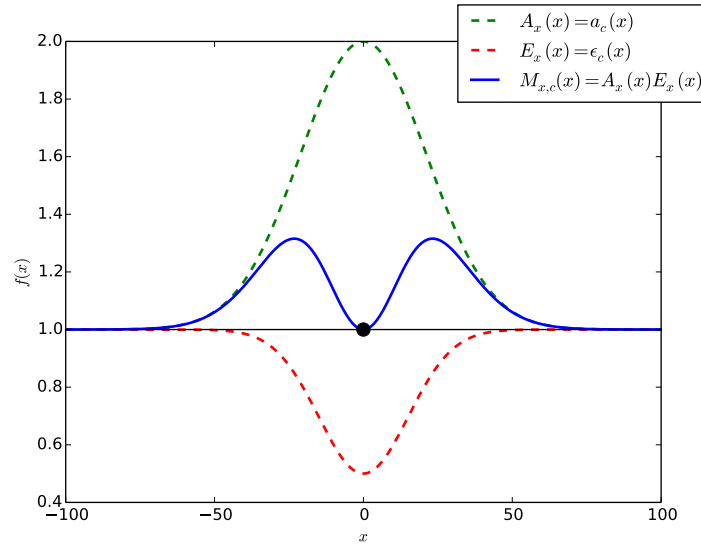


Figure 5.3.2: The effect of an adjacent joint a (dot) on the likelihoods of class c in the neighbourhood surrounding joint a .

The effect of a placed joint on the weight image of an adjacent unplaced joint is shown in **Figure 5.3.2**. The figure shows the effect of the function M_c as viewed along the x -dimension. The two peaks of the blue line show that the likelihood values in the neighbourhood of the placed joint in either direction are slightly increased, while the likelihoods at the joint and far away from the joint are unaffected.

Figure 5.3.3 shows what happens if the situation of **Figure 5.3.2** is changed by placing another, non-adjacent joint of similar size in line with the first joint. The likelihoods to the right of the first placed joint remain unchanged, while those to the left

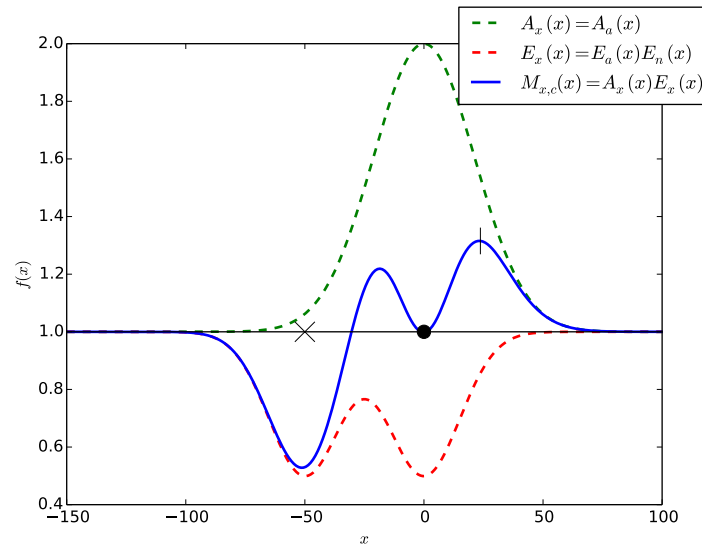


Figure 5.3.3: The effect of an adjacent joint a (dot) and a non-adjacent joint n on the likelihoods of class c in the neighbourhood surrounding the joints. The horizontal line in the sketch indicates the maximum of $M_c(x)$.

are slightly decreased. The joint is thus encouraged to be placed to the right of the adjacent joint. Furthermore, the likelihood that the joint will be placed in the region of the non-adjacent joint is decreased.

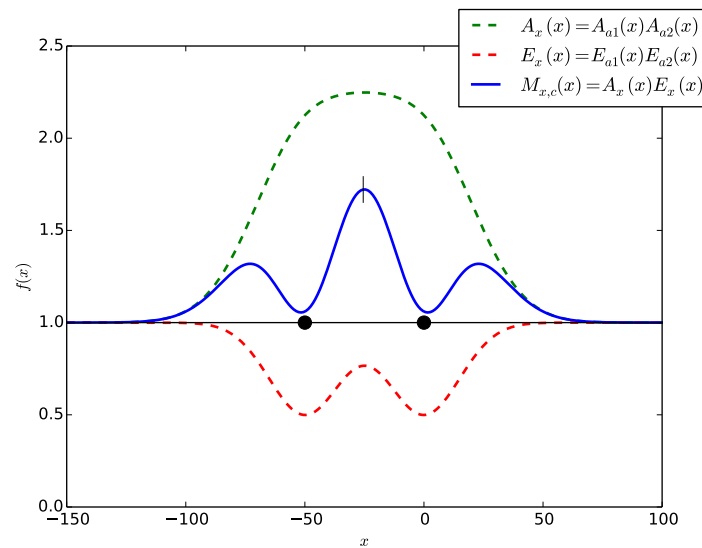


Figure 5.3.4: The effect of two adjacent joints a_1 and a_2 (dots) on the likelihoods of class c in the neighbourhood surrounding the joints. The horizontal line in the sketch indicates the maximum of $M_c(x)$.

Figure 5.3.4 shows how the previous example will change if the non-adjacent joint is changed into an adjacent joint. This situation is typically seen when the two placed joints are connected with the unplaced joint, such as the knuckle of a finger being con-

nected to both the base of the finger and the second joint from the finger tip. In this case the most likely place the joint will be located is between these two joints and $M_c(\mathbf{x})$ ensures that the joint's likelihoods are considerably increased in this region. This is especially important when the per pixel classification at this region is weak because of partial occlusion of the joint.

5.3.4 Final Joint Coordinates

The final step of the joint estimation system is to convert the 2D depth image coordinates of the joints to 3D projection coordinates. This is trivial for most of the joints, since the 2D image coordinates can simply be extended to 3D coordinates using the depth value at the image coordinates as the value of the third dimension. Note that the depth value is actually that of the skin surface around the joint. This does not however concern us, firstly because the skin depth around the joints are uniform across the hand for the views we are interested in and secondly because we are interested only in the shape of the skeleton as opposed to the true location. We can thus use the depth values as is without applying a learned depth offset to find the true location of the joints.

Estimating the 3D coordinates of the occluded joints represents more of a challenge. Assuming the mean shift procedure described in the previous section was able to place the joint near its true 2D depth image coordinates, the depth of the joint might still be unreliable. There are two cases of occlusion to consider. The first case involves the occlusion of a joint by another non-adjacent joint. In this case, we calculate the depth of the joint as the average between its adjacent joints. The second case is found when a joint is occluded by an adjacent joint. In this case, we assume the occluded joint is perpendicular to the adjacent joint and simply set the occluded joint's depth to that of the adjacent joint plus one depth unit (2cm in the case of the Kinect).

5.4 Testing

This section describes the tests performed to verify the performance of the three Joint Estimators described in this chapter, namely the Centre of Gravity, Mean-Shift and Reservation estimators. Tests were performed to determine the speed, accuracy and consistency of each estimator. The first subsection of this section describes the experimental setup, while the subsequent sections describe the tests performed and discuss the results.

5.4.1 Experimental Setup

The test described in this section used the same synthetically generated depth images used to test the Region Classifier and described in **Section 4.3**, specifically those generated for Testing Set A. The testing images were classified using the RDF16_AT, RDF16_AL, RDF16_BT and RDF16_BL region classifiers described in **Section 4.3**. The resulting pixel label probabilities were used to test the Joint Estimators.

A Gold Standard joint set was created in order to serve as a baseline for comparing the joint estimators. The baseline joints set was created by processing the labelled depth images with the Centre of Gravity joint estimator. There are two reasons for using this approximation as opposed to extracting the true joint coordinates from the hand model. Firstly, the 3D modelling program used (Blender) does not provide functions for extracting the image coordinates of points in world space. Secondly, there is no functionality for determining if a joint is occluded from the 3D model, while the labelled depth images can be used to determine if a joint is occluded by finding the total labels present in the image for a particular region. These problems can be overcome, though the retrieval of approximated joints is quick to implement and provides a reliable baseline.

The effect of noise on the system was also measured, by creating variations of Testing Set A where noise was applied to the depth images. We decided to use a pixel shuffling algorithm to apply noise to the depth images, in order to simulate the ragged edges and rough surfaces seen on the depth images retrieved from the Kinect. The algorithm selected foreground pixels at random and swapped the pixel with another pixel within a 4 pixel radius. Two examples of depth images with noise applied are shown in **Figure 5.4.1**.

The shuffling algorithm also emulates the effects of quantisation noise on the depth images. Quantisation noise results in the surface of the hand not being smooth. The shuffling algorithm inherently causes the surface of the hand to become rough. We could however introduce a synthetic quantisation of the pixels to be more faithful to the original noise component, but expect to see little difference in the test results.

The pose recognition system assumes that only pixels belonging to the hand is given as input, as stated in **Section 2.3.4**. Our hand tracking component is able (in most cases) to successfully extract these pixels, which means we technically do not have to test for this noise. The shuffling algorithm does however introduce a certain component of this noise, as can be seen by the "blurred" hand edges, where it is not

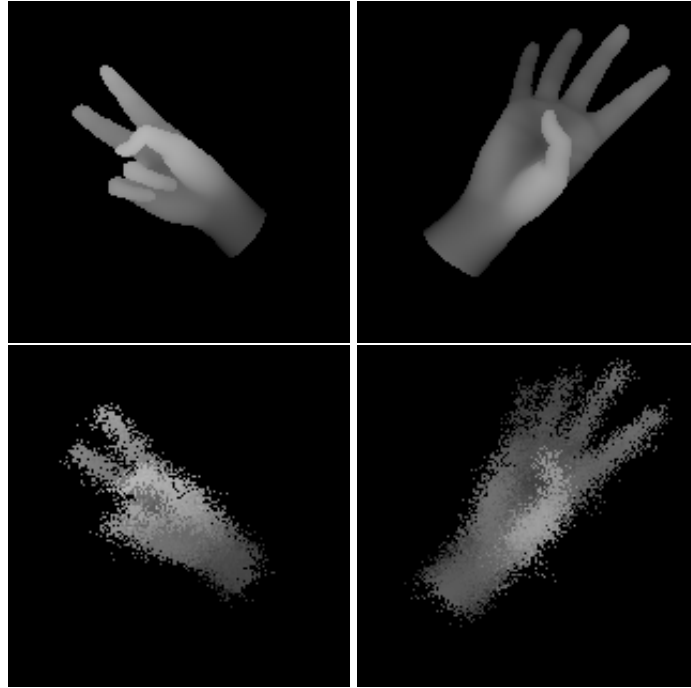


Figure 5.4.1: Examples of depth images with noise applied. The top row shows the original depth images, while the bottom row shows the images with noise applied.

clear whether a pixel actually belongs to the hand or not (it originally did, but after the shuffling algorithm is applied is might fall outside the original hand region).

5.4.2 Joint Placement

The Joint Placement tests evaluated whether the joint estimators were able to find the hand joints and place them on the surface of the hand. These preliminary tests mainly serve to reveal various characteristics of the estimators and to determine whether the tested algorithms were not fundamentally flawed in any way. A more standard way of evaluating the accuracy of the joint estimators is shown in the next section.

The first test counted the number of joints not found during joint estimation. Joints are usually not found when the hand region corresponding to a joint is obscured or because of a particularly poor classification from the Region Classifier. **Figure 5.4.2** shows the number of joints found missing for each of the fingers while estimating the joints from Testing Set A. The graph shows that the hard decision Centre of Gravity joint estimator performed the worst, while the soft decision Mean-Shift and Reservation always find an estimate for the joints. This is due to the fact that with the soft decision classifiers, there will always be partial evidence of a hand region, in the form of probabilities, while the hard decision Centre of Gravity classifier might have no evidence.

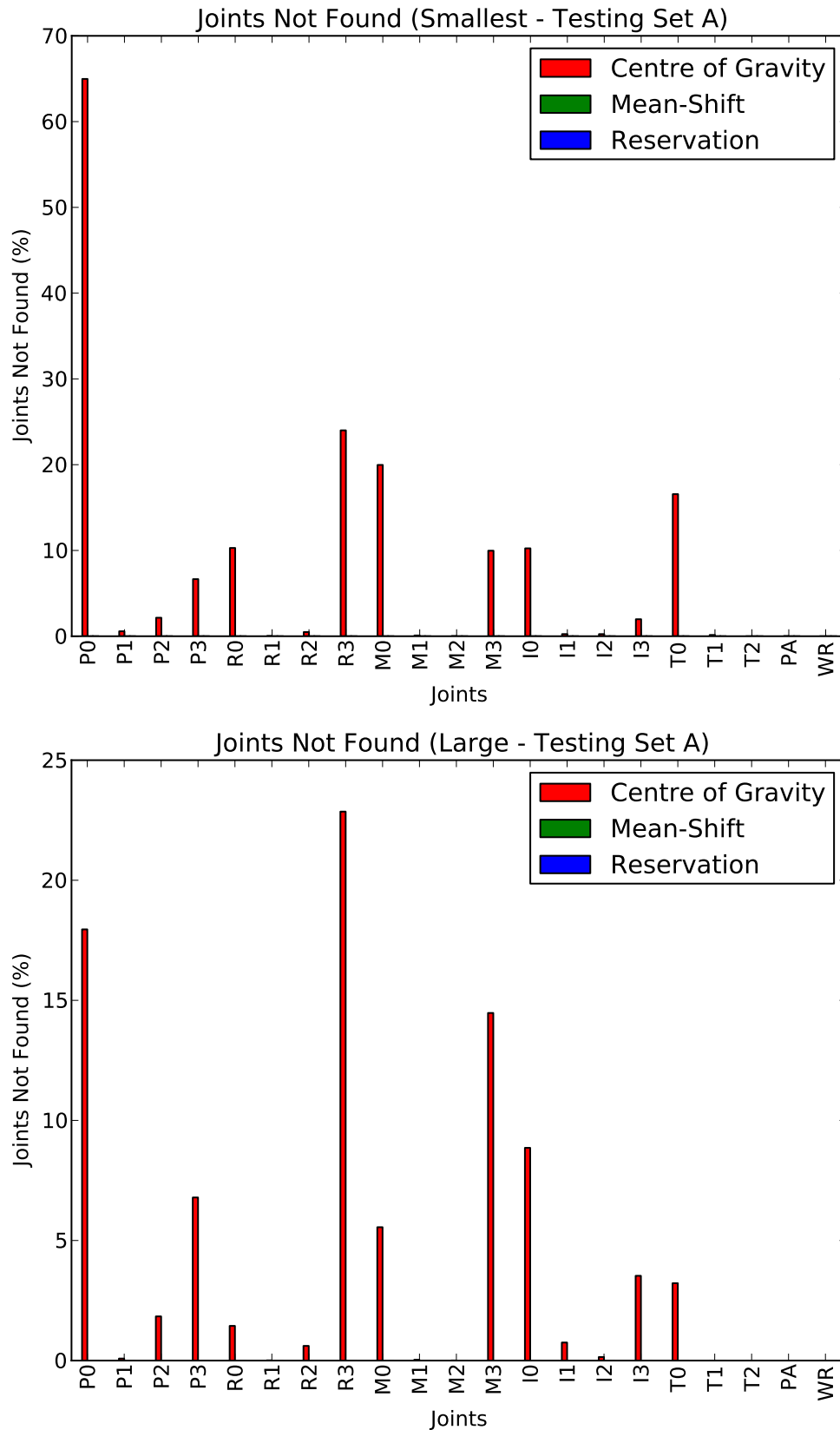


Figure 5.4.2: Graphs of the number of joints found missing using various region classifier and joint estimator combinations. The top graph shows the results when using region classifier RDF16_AT, while the bottom graph shows RDF16_AL. (Note difference in y-axis scale.)

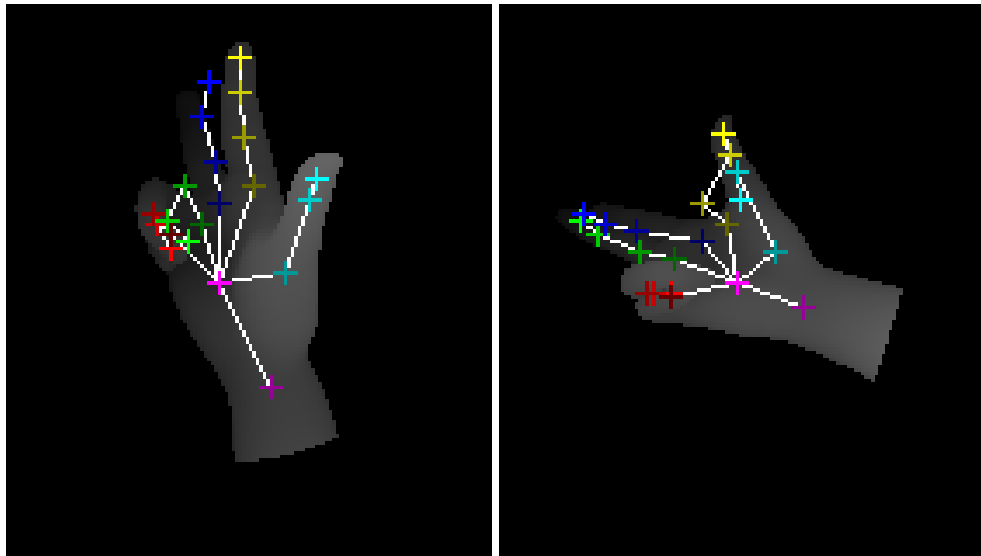


Figure 5.4.3: Examples of poses where joints are placed on the background. (Misplaced Joints – Left: R2, M0, M1; Right: M0, I2)

Table of Joint Placement Results		
Joint Estimator	Total Joints Not Found (%)	Total Joints Misplaced (%)
Centre of Gravity	8.03	11.51
Mean-Shift	0.0	2.55
Reservation	0.0	1.31

Table 5.4.1: Table showing the total joints lost or misplaced by the joint estimators.

The second test determined how frequently the joints estimators placed joints on background pixels as illustrated by **Figure 5.4.3**. **Figure 5.4.4** shows how many joints per finger the joint estimators erroneously placed on background pixels. Note that these results include joints that were not found by the joint estimator. Also, the test does not give an indication of how many joints are erroneously placed on the wrong region of the hand, only focusing on whether a joint was placed on the hand at all. The results are similar to that of the previous test, where the Centre of Gravity estimator performed poorly compared to the more complex Mean-Shift and Reservation estimators.

The results shown in **Figure 5.4.2** and **Figure 5.4.4** indicate that the wrist, palm and lower thumb regions are generally the best estimated joints using these test measurements. This is expected, considering the large size of these regions and low chance of full occlusions. The finger tips (*0) and knuckle joints (*3) are the least reliable joints. The finger tips are generally small and poorly classified as shown in the tests of **Section 4.3**, which makes joint estimation difficult. The knuckle joints suffer from occlusion problems, caused by closed hand poses where one or more of the fingers are bent, causing the upper part of a finger to occlude the knuckle.

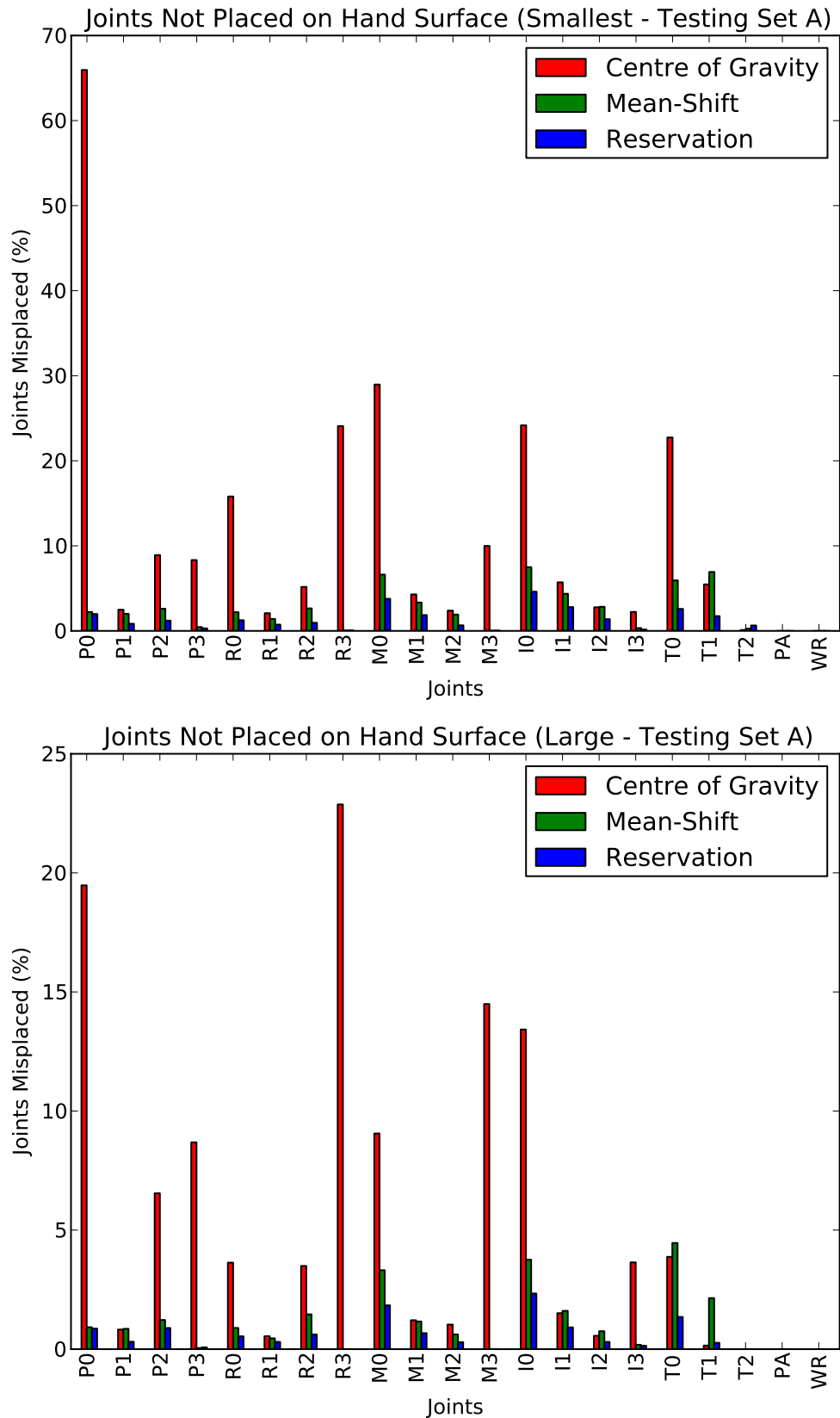


Figure 5.4.4: Graphs of the number of joints misplaced using various region classifier and joint estimator combinations. The top graph shows the results when using region classifier RDF16_AT, while the bottom graph shows RDF16_AL. (Note difference in y-axis scale.)

Table 5.4.1 shows the total missing and incorrectly placed joints for each joint estimator. The results show that the probability-based Mean Shift and Reservation joint estimators performed similarly and performed better than the hard-decision Centre of Gravity joint estimator.

5.4.3 Accuracy of Joint Estimators

The next set of tests provides a more standard way of measuring the accuracy of the joint estimators. The classified depth images of Testing Set A were again used to test the joint estimators. The Gold Standard joint set is used again as a baseline for comparing the joint estimators.

The accuracy tests measured the 2D image vector distance between the estimated joints and the Gold Standard joint set. The 2D image distances were used instead of the 3D world distances to avoid penalising joints not placed on the hand surface, where the large difference between foreground and background depth would skew the results. Joints were placed at the centre of the hand if the joint estimator was unable to find the joints for these measurements. As shown by the joint placement tests in the previous section, this only influences the results of the Centre of Gravity estimator.

Figure 5.4.5 shows the joint distances between the estimated and baseline joint sets. The graphs show that all estimators are able to place most joints within 4 pixels of the baseline joints. The joints that prove to be the hardest to estimate correctly are the finger tips (*0) and the knuckle joints (*3). This is again due to the small size of the finger tips and the regular occlusion of the knuckle joints. **Figure 5.4.6** shows the tests performed on the noisy testing data, which show an overall decrease in accuracy. The Mean-Shift and Reservation estimators generally perform better than the Centre of Gravity estimator.

5.4.4 Consistency of Joint Estimators

A joint estimator that is able to consistently place joints in a small region around the true joint positions can be useful in a pose estimation system, even if the accuracy of the estimator is average. This is true if the pose classifier is able to recognise the consistent error as a feature of the estimated sets and use it to classify the final pose.

The consistency tests measure the standard deviation of the vector distances from the previous accuracy test. This gives a measure of how consistent the joint estimators are when placing individual joints, which can be used as an indication of how well a

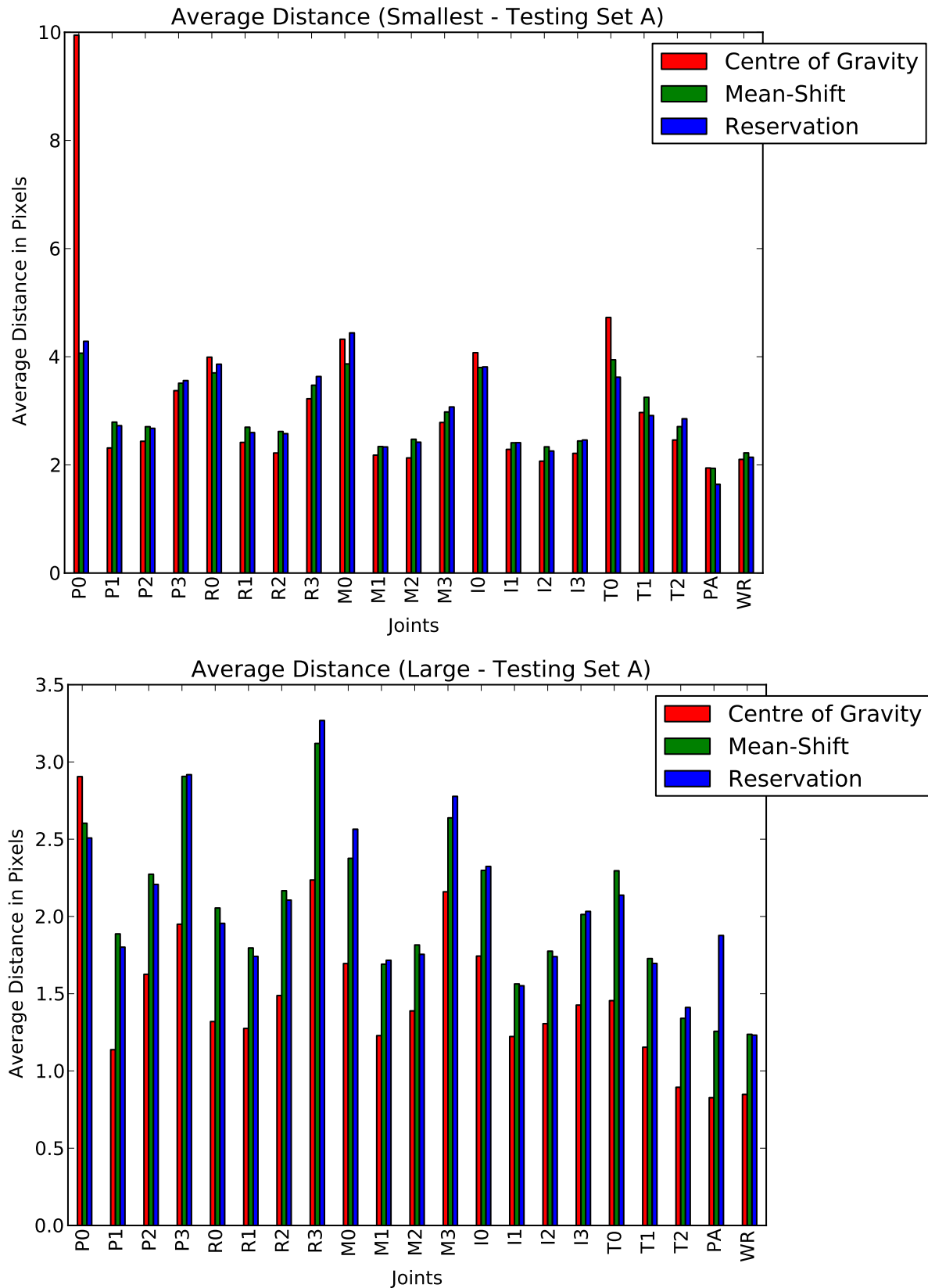


Figure 5.4.5: Average distances between the estimated joint sets from the three joint estimators and the GS Centre of Gravity joint set. The top graph shows the results when using region classifier RDF16_AT to classify the depth images, while the bottom graph shows RDF16_AL.

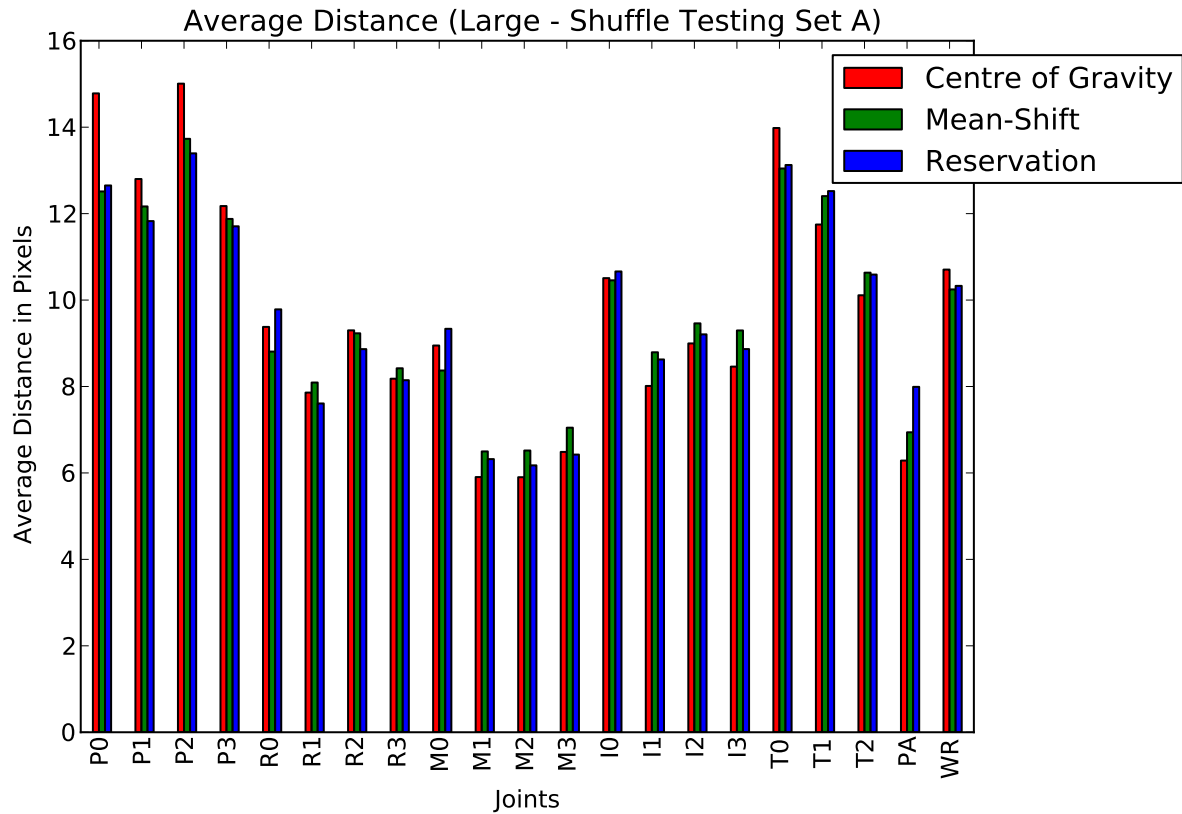


Figure 5.4.6: Average distances between the joint sets estimated from the noise testing set and the Gold Standard joint set.

joint estimator will work in a pose recognition system.

The first test performed measured the standard deviation using the Gold Standard joint set as the baseline. The results for this test are shown in **Figure 5.4.7**. It can be seen that the Mean-Shift and Reservation estimators are generally more consistent than the COG estimator when estimating the location of the joints. Similar to the distance measurements, the finger tip and knuckle joints (*0 and *3), are the most inconsistently estimated joints.

The second test measured the effect of applying noise to the system using the shuffle algorithm. The results of the test are shown in **Figure 5.4.8**. The Mean-Shift joint estimator performed the best in general, yet the Centre of Gravity estimator did not perform as poorly as expected.

The results for the accuracy and consistency tests are summarised in **Table 5.4.2**. The normal Mean-Shift algorithm performed the best, followed by the modified Reservation estimator. The Centre of Gravity estimator performed better than expected and was able to produce results similar to that of the other classifiers.

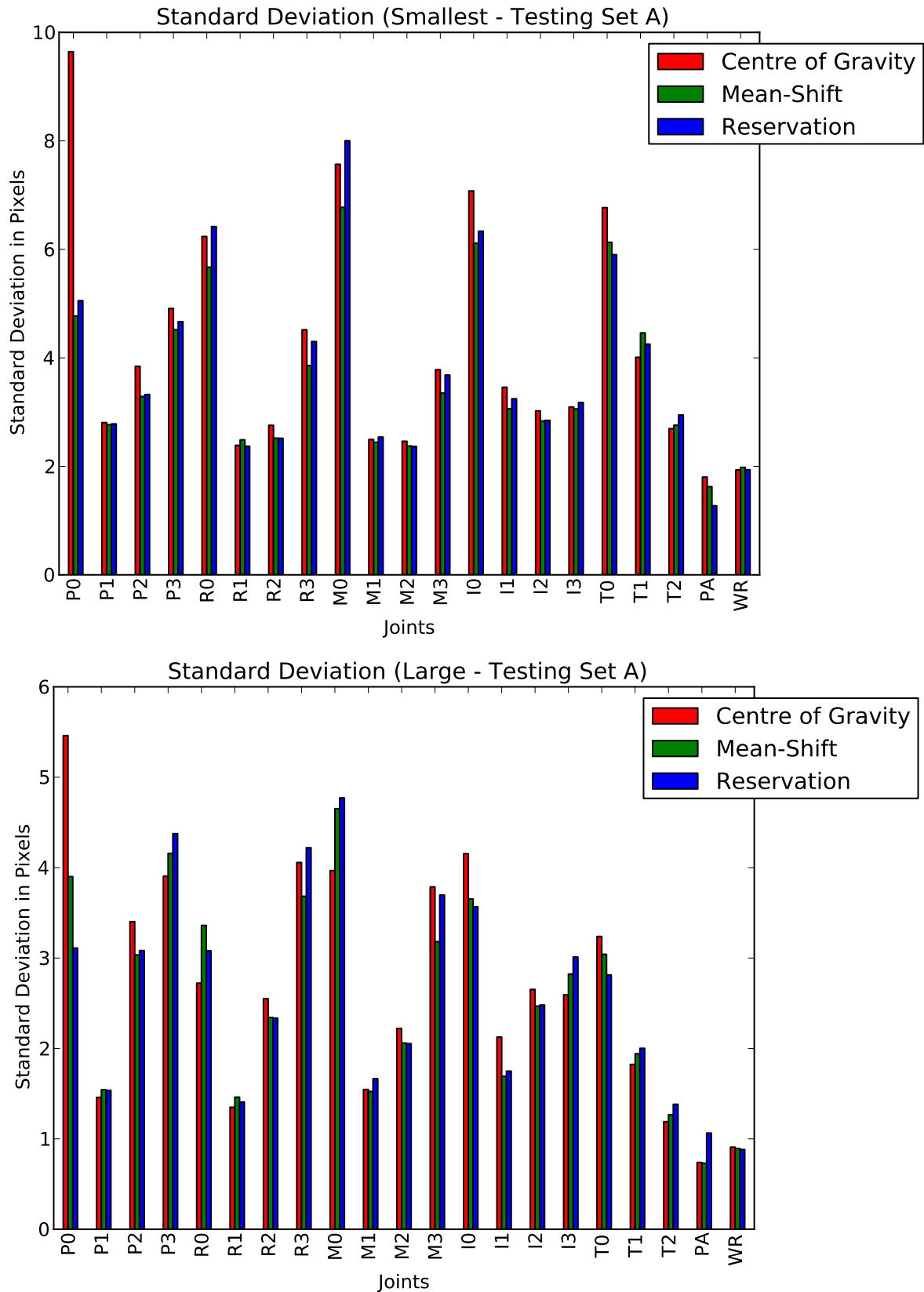


Figure 5.4.7: Standard deviation between the estimated joint sets from the three joint estimators and the GS Centre of Gravity joint set. The top graph shows the results when using region classifier RDF16_AT to classify the depth images, while the bottom graph shows RDF16_AL.

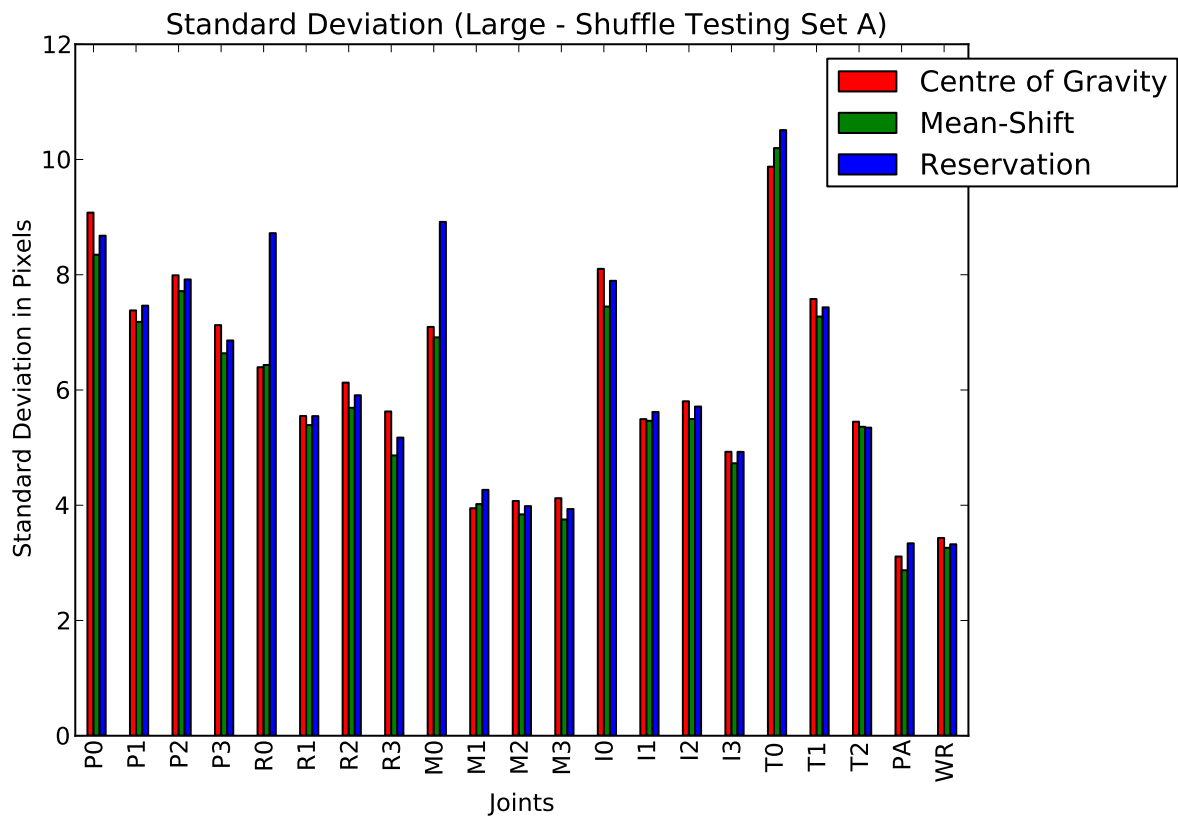


Figure 5.4.8: Standard deviation between the joint sets estimated from the noise testing set and the Gold Standard joint set.

Table of Accuracy and Consistency Results			
	Centre of Gravity	Mean-Shift	Reservation
Distance (Smallest)	3.15	2.97	2.97
Distance (Large)	1.49	2.04	2.06
Distance (Large - Noise)	9.90	9.72	9.70
Standard Deviation (Smallest)	4.16	3.66	3.81
Standard Deviation (Large)	2.66	2.54	2.56
Standard Deviation (Large - Noise)	6.27	5.84	6.26

Table 5.4.2: Table summarising the accuracy and consistency results of the joint estimators. The best results for each test are marked in bold.

5.4.5 Speed of Joint Estimators

The final set of tests determines the speed of each of the three joint estimators when used together with the Region Classifier. These tests were used to determine which combinations of region classifiers and joint estimators give a system that can extract joints in real-time. Region Classifiers trained using the Smallest and Large subsets of Training Set A were used for testing, with forest sizes of 1, 2, 4, 8 and 16. The results are shown in **Figure 5.4.9**.

It can be seen from the test results that the forest size of the region classifier has the greatest effect on the joint estimation time. The Centre of Gravity was the fastest estimator and had little effect on the time taken to estimate joints. The Reservation classifier performed the slowest, and could not estimate joints in real-time in half of the tests performed. The Mean-Shift estimator, while slower than the COG estimator, was able to estimate joints in real-time for all but 2 of the 10 tests, which is due to the slow speed of the 16-tree classifiers.

5.5 Summary

This chapter discussed the various Joint Estimators implemented for this project. In **Section 5.2** we discussed the theory of the Mean-Shift algorithm, which is used to find the modes of an unknown probability density function. **Section 5.3** described the system implementation of the Mean-Shift algorithm to find the location of the joints from the pixel probabilities calculated using the Region Classifier, including our Reservation variation on the Mean-Shift algorithm which uses various heuristics specific to our problem to estimate the joints.

Section 5.4 showed and discussed the results of various tests performed to determine the performance of the joint estimators. The tests showed that the Centre of Gravity joint estimator is the fastest and performed well when the Region Classifier accurately classified a given depth image. The Mean-Shift joint estimators, though slower, are preferred when noise is added to the system and the final classification of the depth image is less accurate.

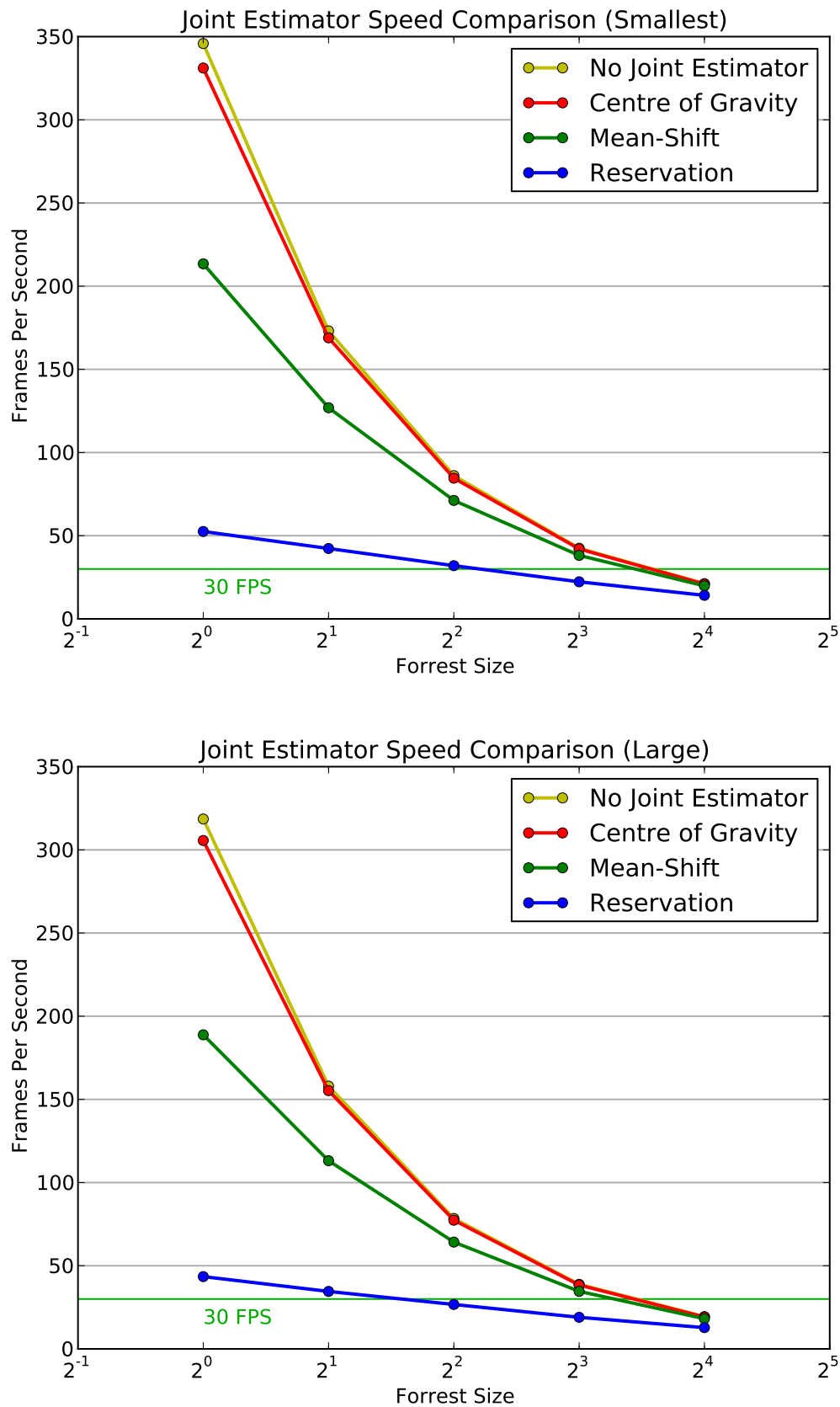


Figure 5.4.9: The speed measurements for various combinations of region classifiers and joint estimators. The top graph shows the speed measurements for region classifiers trained using the Smallest subset of Training Set A, while the bottom graph shows the measurements for the Large subset.

Chapter 6

Support Vector Machines

This chapter discusses the theory behind the classifiers used for pose classification, namely Support Vector Machines (SVMs). The two largest sources of information used to derive the theoretical basis for SVMs were [41] and [47], though the numerous published guides [48, 49, 50] proved to be more accessible starting points for those unfamiliar with SVMs.

Our system is designed to accurately classify 17 different poses of which several poses are closely related, such as the “open” and “closed” poses. The problem space thus consists of multiple classes which are not linearly separable. This chapter first discusses the basic SVM classifier, followed by explanations of how the SVM can be extended to handle multiclass and nonlinear problems.

6.1 Basic Support Vector Machine

The unmodified SVM classifier operates on linearly separable data that consists of only two classes. This section describes the workings of the unmodified SVM. Subsequent sections will describe the various modifications made to the algorithm to produce multiclass, nonlinear classifiers, which are needed to accurately classify hand poses for our system.

The following is a summary of the theoretical development of the SVM discussed in this section:

1. A basic discriminative function using a hyperplane is defined as the basis of an

SVM classifier. The discriminative function can be written mathematically as

$$y(\mathbf{x}) = \begin{cases} 1, & \mathbf{w}^T \mathbf{x} \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

where \mathbf{w} is a weight vector and \mathbf{x} is a data point to be classified using label $y(\mathbf{x})$.

2. The Support Vector Machine classifier is defined as the discriminative function using the hyperplane which has the largest error margin between the decision boundary and the nearest data points. Various constraints are placed on the problem in such a way that the equation of the boundary to the error margin is written as

$$\left| \mathbf{w}^T \mathbf{x}_s + b \right| = 1, \quad \forall s \in \left[1, \dots, N_s \right]$$

where b is w_0 extracted from \mathbf{w} and is known as the bias. The Support Vectors \mathbf{x}_s are the N_s points which lie on the error margin boundary and are closest to the decision boundary.

3. The width of the error margin is shown to be equal to $\frac{2}{\|\mathbf{w}\|}$.
4. An optimisation problem is formulated to find the values of \mathbf{w} and b which maximise the width of the error margin. The problem proves to be difficult to solve for various reasons and is manipulated into an equivalent convex optimisation problem which is easier to solve. The final problem is:

$$\begin{aligned} & \text{minimise} && \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ & \text{subject to} && y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad i = \left[1, \dots, N \right] \end{aligned}$$

5. The above optimisation problem is further manipulated into an equivalent form which is easier to solve by finding the Wolfe Dual Representation of the convex optimisation using Lagrange functions and enforcing the Karush-Kuhn-Tucker necessary conditions to ensure that the local minimiser finds the optimal solution.

The final optimising problem in terms of the Lagrangian Multipliers λ is:

$$\begin{aligned} \max_{\lambda} \quad & \left(\sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{j=1}^N \sum_{i=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_i \mathbf{x}_j^T \right) \\ \text{subject to} \quad & \sum_{i=1}^N \lambda_i y_i = 0 \\ & \lambda \geq \mathbf{0} \end{aligned}$$

6. The above optimisation is solved for λ , which is then used to find the optimal values of \mathbf{w} and b . These values are then used for the final 2-class classifier:

$$y(\mathbf{x}) = \begin{cases} 1, & \mathbf{w}^T \mathbf{x} + b \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

6.1.1 Linear Discriminant Function

The SVM algorithm is at its core a linear discriminant classifier. The SVM, similar to a perceptron, is a simple binary classifier used to classify two-class linearly separable problems using a hyperplane. The hyperplane is of the form

$$p(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \quad (6.1.1)$$

where \mathbf{x} is the homogeneous feature vector and \mathbf{w} is a weight vector, both having d dimensions. The hyperplane can be used to label data points using the following discriminant function:

$$y(\mathbf{x}) = \begin{cases} 1, & p(x) \geq 0 \\ -1, & \text{otherwise} \end{cases} \quad (6.1.2)$$

Equation (6.1.2) assigns a label of 1 for data points that fall above and on the decision boundary at $p(\mathbf{x}) = 0$, while a value of -1 are assigned to all other data points.

The discriminant function uses the hyperplane to divide the given data set into two sections, where each section is associated with one of the classes. **Figure 6.1.1** illustrates the working of the discriminant function for a 2D feature space. The top-left image marked **S** shows the unlabelled feature space, which consists of two linearly separable classes. The other images show three different hyperplanes which are marked **A**, **B**, **C**. Each hyperplane is visualised using a line indicating the decision boundary. The blue and red points indicate how the data points are labelled using the given hyperplane.

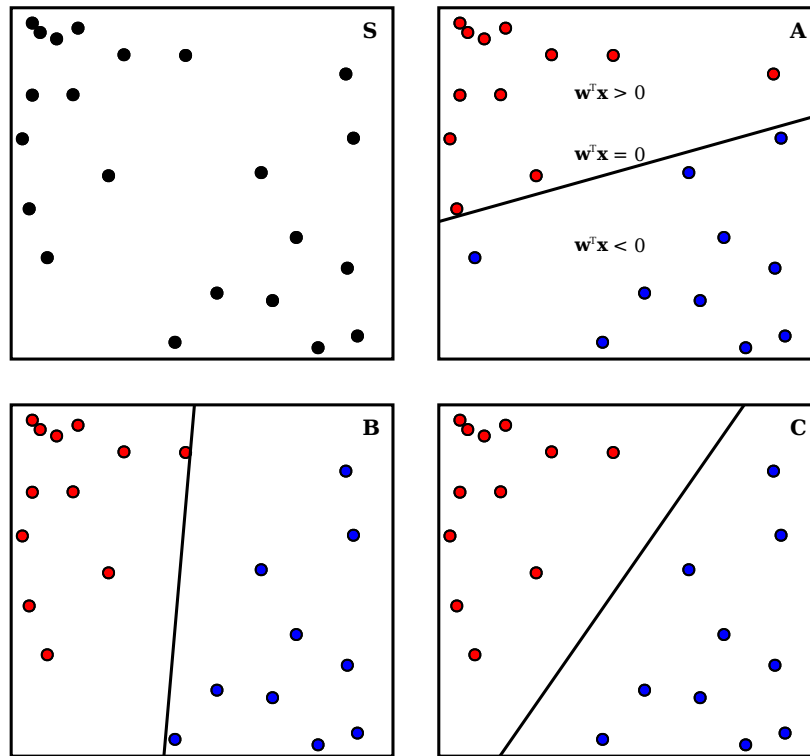


Figure 6.1.1: Illustration of how the discriminant function classifies a 2D linearly separable two-class problem using a hyperplane. Figure S shows the original unlabelled data set. Figures A, B and C show how different hyperplanes will label the data set. Notice that the hyperplanes of B and C give the same classification, but C leaves more room for error.

Hyperplane **A** is clearly a poor choice to use for classifying the given data set, while **B** and **C** are both able to classify the data set perfectly. Hyperplanes **B** and **C** are just two examples of many possible hyperplanes which can be used to classify the data with perfect accuracy. This leads to the question of whether one of these hyperplanes is better than the other. Even though Hyperplanes **B** and **C** perform identically for the given data set, it is clear that **C** should perform better in general, given that it leaves a larger error margin between the outermost class data points and the decision plane. Finding this hyperplane which maximises the error margin is the main goal of SVMs. The next sections discuss how the hyperplane which has the maximum error margin can be found.

6.1.2 Preliminary Equations and Constraints

It is necessary to first define a few equations and constraints related to the hyperplane. We can define a d -dimensional hyperplane using a homogeneous coordinate system as

$$\begin{aligned} \begin{bmatrix} w_1 & \cdots & w_d & w_0 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_d \\ 1 \end{bmatrix} &= 0 \\ \mathbf{w}'^T \mathbf{x} &= 0 \end{aligned} \quad (6.1.3)$$

The term associated with w_0 and $x_0 = 1$ is separated from \mathbf{w}' and renamed to b for the bias. This separation simplifies the mathematics later in the chapter and gives the following equation for the hyperplane:

$$\begin{aligned} \begin{bmatrix} w_1 & \cdots & w_d \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} + b &= 0 \\ \mathbf{w}^T \mathbf{x} + b &= 0 \end{aligned} \quad (6.1.4)$$

Substituting any point \mathbf{x} into (6.1.4) gives the function distance between the point and the hyperplane.

Notice that the left side of (6.1.4) can be multiplied with any scalar factor without affecting the final decision boundary. The equation can be made scale invariant by normalising it such that the function distances between the plane and the nearest points both above and below the plane are 1:

$$|\mathbf{w}^T \mathbf{s}_i + b| = 1, \quad \forall i \in [1, \dots, N_s] \quad (6.1.5)$$

In the above equation, \mathbf{s}_i represents the N_s support vectors. The support vectors are defined as the points which lie on the boundary of the error margin and are the closest to the hyperplane. The support vectors and error margins at 1 and -1 are shown in **Figure 6.1.2**. The next section will discuss finding the plane which gives the largest error margin and show that the total error margin is $\frac{2}{\|\mathbf{w}\|}$ if (6.1.5) is satisfied.

6.1.3 Error Margin Formulation

As mentioned previously, any point \mathbf{x}_i can be substituted into $\mathbf{w}^T \mathbf{x} + b$ to retrieve the function distance between the point and the plane. These function distances can however not be used to compare the distances between a given point and several hyperplanes, as each distance measurement is dependant on the function used to describe the plane. The Euclidean distance can be used instead as a more standard way of measuring the distance between a point and several planes. This section shows how the formula for this Euclidean distance is derived.

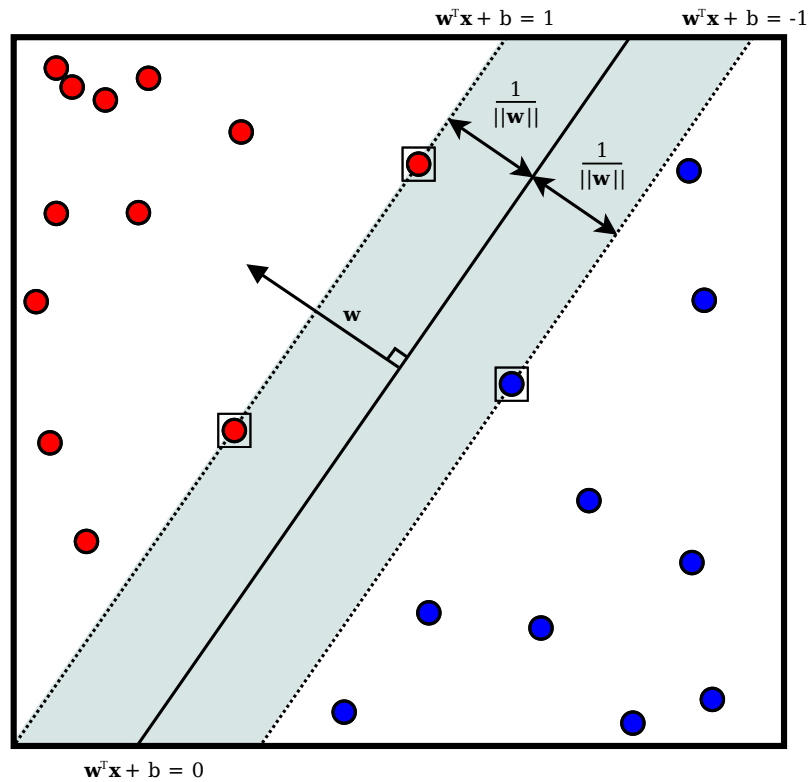


Figure 6.1.2: Illustration of how the hyperplane of the discriminant function classifies a data set. The points enclosed in a square are the Support Vectors, which determine the size of the error margin as shown in blue.

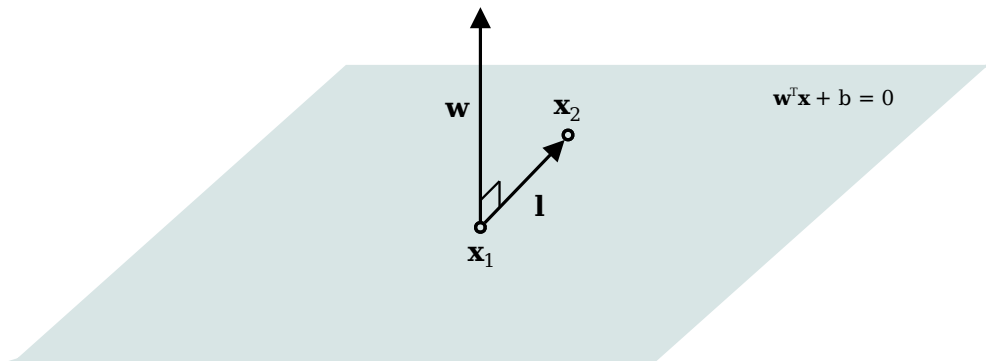


Figure 6.1.3: A hyperplane and the corresponding w vector. The points labelled x_1 and x_2 lie on the plane and are connected by the vector l .

It can be shown that the vector represented by w is perpendicular to the hyperplane described by (6.1.4). Taking any two points x_1 and x_2 which lie on the hyperplane, as shown in Figure 6.1.3, and substituting them in (6.1.4) give:

$$w^T x_1 + b = 0$$

$$w^T x_2 + b = 0$$

Subtracting the above equations from each other:

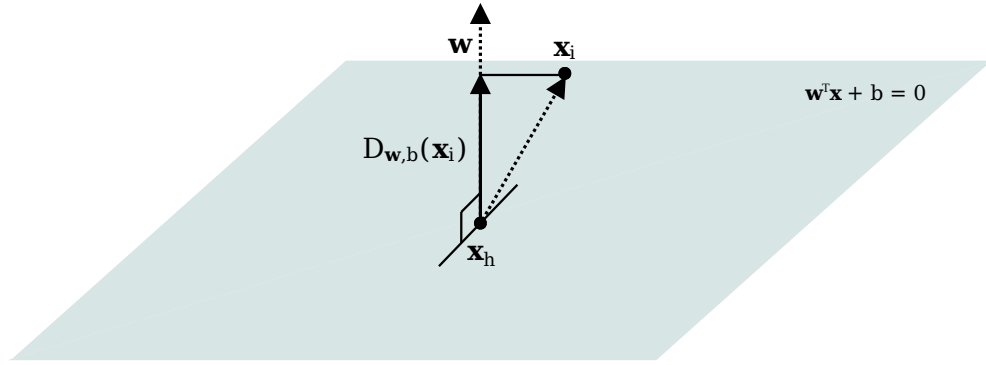


Figure 6.1.4: Illustration of the vectors that are used to calculate the Euclidean distance between any point and the hyperplane.

$$\begin{aligned}
 \mathbf{w}^T \mathbf{x}_2 + b - \mathbf{w}^T \mathbf{x}_1 - b &= 0 \\
 \mathbf{w}^T \mathbf{x}_2 - \mathbf{w}^T \mathbf{x}_1 &= 0 \\
 \mathbf{w}^T (\mathbf{x}_2 - \mathbf{x}_1) &= 0 \\
 \mathbf{w}^T \mathbf{l} &= 0
 \end{aligned}$$

The vector \mathbf{l} is the vector joining \mathbf{x}_1 and \mathbf{x}_2 and lies on the plane. The above proof shows that the inner product between \mathbf{w} and \mathbf{l} is zero, making the two vectors perpendicular and proving that \mathbf{w} is always perpendicular to the hyperplane.

We can thus find the euclidean distance between any point \mathbf{x}_i and the hyperplane by calculating the projection of the point onto \mathbf{w} :

$$D_{\mathbf{w},b}(\mathbf{x}_i) = \left| \hat{\mathbf{w}}^T (\mathbf{x}_i - \mathbf{x}_h) \right| \quad (6.1.6)$$

The points \mathbf{x}_i and \mathbf{x}_h are shown in **Figure 6.1.4**, where \mathbf{x}_h is the point where \mathbf{w} intersects the hyperplane. The unit vector $\hat{\mathbf{w}}$ is calculated as

$$\hat{\mathbf{w}} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

where $\|\mathbf{w}\|$ is the vector norm. The absolute value ensures that the distance is always positive, independent of the direction of the vector \mathbf{w} . We can rewrite (6.1.6) by substituting $\hat{\mathbf{w}}$ and performing a few manipulations:

$$\begin{aligned}
 D_{\mathbf{w},b}(\mathbf{x}_i) &= \left| \hat{\mathbf{w}}^T (\mathbf{x}_i - \mathbf{x}_h) \right| \\
 &= \left| \frac{\mathbf{w}^T}{\|\mathbf{w}\|} (\mathbf{x}_i - \mathbf{x}_h) \right| \\
 &= \frac{1}{\|\mathbf{w}\|} \left| \mathbf{w}^T \mathbf{x}_i - \mathbf{w}^T \mathbf{x}_h \right| \\
 &= \frac{1}{\|\mathbf{w}\|} \left| \mathbf{w}^T \mathbf{x}_i + b - \mathbf{w}^T \mathbf{x}_h - b \right| \\
 &= \frac{1}{\|\mathbf{w}\|} \left| (\mathbf{w}^T \mathbf{x}_i + b) - (\mathbf{w}^T \mathbf{x}_h + b) \right|
 \end{aligned}$$

Notice that the second term of the absolute value equates to zero, since \mathbf{x}_h lies on

the hyperplane. Furthermore, if \mathbf{x}_i is one of the support vectors \mathbf{s}_i , then the left term equates to 1 according to normalisation constraints of (6.1.5). The distance between a support vector \mathbf{s}_i and the hyperplane described by \mathbf{w} and b can thus be written as:

$$D_{\mathbf{w},b}(\mathbf{s}_i) = \frac{1}{\|\mathbf{w}\|} \quad (6.1.7)$$

The total width of the error margin is thus $\frac{2}{\|\mathbf{w}\|}$ if (6.1.5) holds true. The next section will show how an optimisation problem can be formulated to find the vector \mathbf{w} and scalar b which maximise the margin.

6.1.4 Optimising for Largest Margin

The optimisation problem for finding \mathbf{w} that leads to the largest error margin is:

$$\text{maximise} \quad \frac{2}{\|\mathbf{w}\|} \quad (6.1.8)$$

$$\text{subject to} \quad \min_{i=1 \dots N} |\mathbf{w}^T \mathbf{x}_i + b| = 1 \quad (6.1.9)$$

There are however two problems with the above formulation. Firstly, the division by the norm and the square root inside the norm make the optimisation computationally expensive and can potentially lead to division by zero or accuracy errors. The division by the norm can be avoided by rather finding the equivalent minimum of the reciprocal term:

$$\text{maximise} \quad \frac{2}{\|\mathbf{w}\|} \Rightarrow \text{minimise} \quad \frac{\|\mathbf{w}\|}{2}$$

The square root can be avoided by simply minimising the square of the norm:

$$\begin{aligned} \text{minimise} \quad \|\mathbf{w}\| &\Rightarrow \text{minimise} \quad \|\mathbf{w}\|^2 \\ &\Rightarrow \text{minimise} \quad \sum_{i=1}^d w_i^2 \\ &\Rightarrow \text{minimise} \quad \mathbf{w}^T \mathbf{w} \end{aligned}$$

The second problem with the original formulation of the optimisation problem is the minimum function and the absolute value inside the constraints. The absolute value can be removed by multiplying the hyperplane function by the correct class labels $y = \{-1, 1\}$:

$$|\mathbf{w}^T \mathbf{x}_i + b| = y_i(\mathbf{w}^T \mathbf{x}_i + b)$$

The constraint can be reformulated to avoid the minimum function by simply using an inequality which ensures that all points are larger or equal to the expected minimum:

$$\begin{aligned} \min_{i=1\dots N} y_i(\mathbf{w}^T \mathbf{x}_i + b) &= 1 \\ \Rightarrow y_i(\mathbf{w}^T \mathbf{x}_i + b) &\geq 1 \end{aligned} \quad (6.1.10)$$

The new constraint always equates to a positive value and ensures that all of the points lie outside the margin boundaries, with the exception of the support vectors \mathbf{s}_i . The support vectors by the definition of the margin (6.1.5) lie on the margin boundaries located at 1, where the constraint of (6.1.5) was used to formulate (6.1.7) and our original optimisation problem. The new formulation of the optimisation problem is:

$$\text{minimise} \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} \quad (6.1.11)$$

$$\text{subject to} \quad y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \quad i = \left[1, \dots, N \right] \quad (6.1.12)$$

The above equations show a quadratic optimisation problem which is subject to an inequality constraint. This new formulation, while easier to solve than the original formulation, can still prove problematic because of the inequality in the constraint. The next section shows how a combination of Lagrange multipliers, Karush-Kuhn-Tucker and the Wolfe Dual Theorem can be used to remove the inequality and solve the optimisation problem.

6.1.5 Lagrange Formulation

The optimisation problem of (6.1.11) and (6.1.12) can be written as

$$\begin{aligned} \text{minimise} \quad & \mathbf{J}(\mathbf{a}) \\ \text{subject to} \quad & f_i(\mathbf{a}) \geq 0 \quad i = \left[1, \dots, N \right] \end{aligned}$$

where \mathbf{a} is a vector of arguments, $\mathbf{J}(\mathbf{a})$ is our cost function and $f_i(\mathbf{a})$ is our constraints. Furthermore, the problem represents a convex programming problem since

$$\mathbf{J}(\mathbf{a}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} \quad (6.1.13)$$

is a convex function and

$$f_i(\mathbf{a}) = y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \quad (6.1.14)$$

is a concave function with the list of arguments:

$$\mathbf{a} = [\mathbf{w}, b] \quad (6.1.15)$$

The Wolfe Dual Representation [41] states that any convex programming problem is equivalent to an optimisation of the corresponding Lagrangian function

$$\max_{\lambda \geq 0} \mathcal{L}(\mathbf{a}, \lambda) \quad (6.1.16)$$

$$\text{subject to } \frac{\delta}{\delta \mathbf{a}} \mathcal{L}(\mathbf{a}, \lambda) = \mathbf{0} \quad (6.1.17)$$

where λ is a vector of Lagrange Multipliers. The Lagrangian function $\mathcal{L}(\mathbf{a}, \lambda)$ is:

$$\mathcal{L}(\mathbf{a}, \lambda) = \mathbf{J}(\mathbf{a}) - \sum_{i=1}^N \lambda_i f_i(\mathbf{a}) \quad (6.1.18)$$

Substituting (6.1.13) and (6.1.14) into (6.1.18) gives:

$$\mathcal{L}(\mathbf{w}, b, \lambda) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \lambda_i \left[y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 \right] \quad (6.1.19)$$

The optimisation problem of (6.1.11) can be reformulated into an equivalent Lagrange function which satisfies the Karush-Kuhn-Tucker (KKT) conditions. This formulation allows us to find the optimal solution for \mathbf{w} which maximises the error margin, where the KKT conditions need to be satisfied to guarantee that the nonlinear problem has an optimal solution. The KKT conditions are

$$\frac{\delta}{\delta \mathbf{a}} \mathcal{L}(\mathbf{a}, \lambda) = \mathbf{0} \quad (6.1.20)$$

$$\lambda_i \geq 0, \quad i = \left[1, \dots, N \right] \quad (6.1.21)$$

$$\lambda_i f_i(\mathbf{a}) = 0, \quad i = \left[1, \dots, N \right] \quad (6.1.22)$$

Substituting (6.1.15) and (6.1.14) into the above gives:

$$\frac{\delta}{\delta \mathbf{w}} \mathcal{L}(\mathbf{w}, b, \lambda) = \mathbf{0} \quad (6.1.23)$$

$$\frac{\delta}{\delta b} \mathcal{L}(\mathbf{w}, b, \lambda) = 0 \quad (6.1.24)$$

$$\lambda_i \geq 0, \quad i = \begin{bmatrix} 1, & \dots, & N \end{bmatrix} \quad (6.1.25)$$

$$\lambda_i \left[y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \right] = 0, \quad i = \begin{bmatrix} 1, & \dots, & N \end{bmatrix} \quad (6.1.26)$$

The next step is to substitute the Lagrange function in constraints (6.1.23) and (6.1.24) and solve the partial derivatives. Substituting (6.1.19) into (6.1.23) and solving the partial derivative give:

$$\begin{aligned} & \frac{\delta}{\delta \mathbf{w}} \left[\frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \lambda_i \left[y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \right] \right] \\ &= \frac{\delta}{\delta \mathbf{w}} \left[\frac{1}{2} \mathbf{w}^T \mathbf{w} \right] - \sum_{i=1}^N \lambda_i \frac{\delta}{\delta \mathbf{w}} \left[y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \right] \\ &= \mathbf{w} - \sum_{i=1}^N \lambda_i y_i \frac{\delta}{\delta \mathbf{w}} (\mathbf{w}^T \mathbf{x}_i + b) \\ &= \mathbf{w} - \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i = \mathbf{0} \end{aligned}$$

Thus:

$$\mathbf{w} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i \quad (6.1.27)$$

Substituting (6.1.19) into (6.1.24) and solving the partial derivative gives

$$\begin{aligned} & \frac{\delta}{\delta b} \left[\frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \lambda_i \left[y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \right] \right] \\ &= - \sum_{i=1}^N \lambda_i \frac{\delta}{\delta b} \left[y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \right] \\ &= - \sum_{i=1}^N \lambda_i y_i \frac{\delta}{\delta b} (\mathbf{w}^T \mathbf{x}_i + b) \\ &= - \sum_{i=1}^N \lambda_i y_i = 0 \end{aligned}$$

Thus:

$$\sum_{i=1}^N \lambda_i y_i = 0 \quad (6.1.28)$$

The problem of (6.1.11) can now be solved by solving:

$$\begin{aligned} & \text{maximise} \quad \mathcal{L}(\mathbf{w}, b, \boldsymbol{\lambda}) \\ & \text{subject to} \quad \mathbf{w} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i \\ & \quad \sum_{i=1}^N \lambda_i y_i = 0 \\ & \quad \boldsymbol{\lambda} \geq \mathbf{0} \end{aligned} \quad (6.1.29)$$

The above optimisation task has no inequality constraints except for the third constraint, which is simple to enforce. The final step is to remove the weight vector \mathbf{w} by expanding (6.1.19) and substituting (6.1.27) and (6.1.28). Expanding (6.1.19) gives:

$$\begin{aligned} \mathcal{L}(\mathbf{w}, b, \boldsymbol{\lambda}) &= \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \lambda_i \left[y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 \right] \\ &= \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \lambda_i y_i (\mathbf{w}^T \mathbf{x}_i + b) + \sum_{i=1}^N \lambda_i \\ &= \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \lambda_i y_i \mathbf{w}^T \mathbf{x}_i - b \sum_{i=1}^N \lambda_i y_i + \sum_{i=1}^N \lambda_i \end{aligned}$$

Substituting (6.1.27) and (6.1.28) into the above expansion gives:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\lambda}) &= \frac{1}{2} \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i^T \sum_{j=1}^N \lambda_j y_j \mathbf{x}_j - \sum_{j=1}^N \lambda_j y_j \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i^T \mathbf{x}_j - b(0) + \sum_{i=1}^N \lambda_i \\ &= \frac{1}{2} \sum_{j=1}^N \sum_{i=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \sum_{j=1}^N \sum_{i=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^N \lambda_i \\ &= \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{j=1}^N \sum_{i=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \end{aligned}$$

The final optimisation problem is:

$$\max_{\lambda} \left(\sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{j=1}^N \sum_{i=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \right) \quad (6.1.30)$$

$$\text{subject to } \sum_{i=1}^N \lambda_i y_i = 0 \quad (6.1.31)$$

$$\lambda \geq \mathbf{0} \quad (6.1.32)$$

6.1.6 Solving \mathbf{w} and b

The optimisation problem of (6.1.30) is solved using any quadratic programming package able to solve convex problems, such as QuadProg++ [51] for C++ or CVXOPT [52] for Python. In order to use these packages, we first need to rewrite (6.1.30) as a minimum function in matrix form:

$$\max_{\lambda} \left(\frac{1}{2} \lambda^T \mathbf{Q} \lambda + [-1]^T \lambda \right) \quad (6.1.33)$$

where

$$\mathbf{Q} = \begin{bmatrix} y_0 y_0 x_0 x_0 & y_1 y_0 x_1 x_0 & \cdots & y_n y_0 x_n x_0 \\ y_0 y_1 x_0 x_1 & y_1 y_1 x_1 x_1 & \cdots & y_n y_1 x_n x_1 \\ \vdots & \vdots & \ddots & \vdots \\ y_0 y_n x_0 x_n & y_1 y_n x_1 x_n & \cdots & y_n y_n x_n x_n \end{bmatrix}$$

and $-\mathbf{1}$ is a vector of length n filled with -1 . The above problem thus only needs the data points and their true labels as input and gives as output the filled λ vector. A large portion of the entries of the solution for λ will be zeros. This is because of the KKT constraint

$$\lambda_i \left[y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 \right] = 0, \quad i = \left[1, \dots, N \right]$$

which forces λ_i to be zero if the data point \mathbf{x}_i is an interior point, i.e. does not lie on the border of the error margin where $y_i (\mathbf{w}^T \mathbf{x}_i + b) = 1$. All the non-zero entries in λ thus correspond to the support vectors.

Once the solution for λ has been retrieved it becomes a simple matter of substitution to find the values of \mathbf{w} and b which give the largest error margin. The zero entries

of λ allow us to rewrite (6.1.27) as

$$\mathbf{w} = \sum_{i=1}^{N_s} \lambda_i y_i \mathbf{s}_i \quad (6.1.34)$$

where \mathbf{s}_i is one of the N_s support vectors. The value for \mathbf{w} can be found by substituting the values of λ with their corresponding data points and labels into (6.1.34). Once \mathbf{w} has been obtained, b can be calculated by substituting \mathbf{w} and any one of the support vectors into the error margin boundary equation

$$y_i(\mathbf{w}^T \mathbf{x}_s + b) = 1$$

where \mathbf{x}_s is a support vector. Once \mathbf{w} and b have been calculated, any new data point can be classified using (6.1.2). The final values for \mathbf{w} and b can then be used to create the final SVM classifier:

$$y(\mathbf{x}) = \begin{cases} 1, & \mathbf{w}^T \mathbf{x} + b \geq 0 \\ -1, & \text{otherwise} \end{cases} \quad (6.1.35)$$

6.2 Soft Error Margin

The previous section described the basic SVM, which performs well on data sets that are linearly separable. Real world data however is rarely linearly separable and would prove problematic to train using the previous algorithm, which assumes there is a configuration of \mathbf{w} and b that classifies the data perfectly. The previous algorithm can be modified to accommodate data sets where there is a small amount of intersection between classes by introducing the slack variables ξ_i into (6.1.10):

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad (6.2.1)$$

This new formulation allows points to fall inside the margin, known as interior points. The slack variable ξ_i gives an indication of how far away the data point x_i is from the error margin boundary. A point which lies on the error margin has a slack variable $\xi_i = 0$. Interior points that are classified correctly have a slack of $0 < \xi_i \leq 1$, while those that are incorrectly classified have a slack of $\xi_i > 1$. The new slack variables are illustrated in **Figure 6.2.1**.

The introduction of the slack variables changes the optimisation problem from (6.1.11) to:

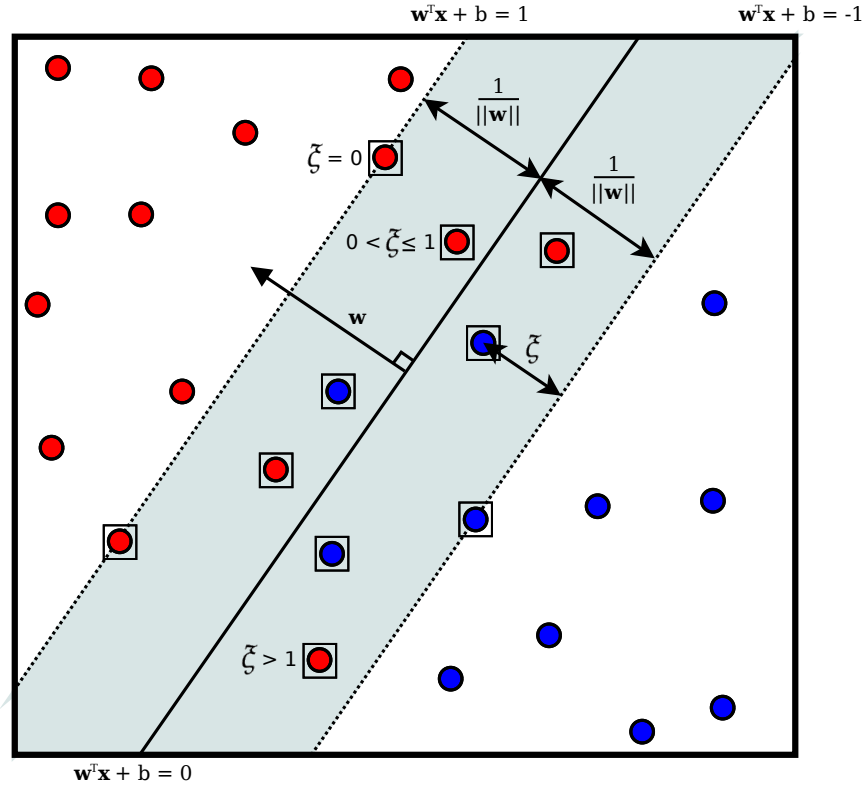


Figure 6.2.1: Illustration of the soft margin and slack variables. The support vectors now include data points inside the margin.

$$\text{minimise} \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^N \xi_i \quad (6.2.2)$$

$$\text{subject to} \quad y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = [1, \dots, N] \quad (6.2.3)$$

$$\xi_i > 0, \quad i = [1, \dots, N] \quad (6.2.4)$$

The scalar multiplier C in the above optimisation task is used to control the influence of the data points inside the margin on the final solution. A small value of C corresponds to a larger margin with more interior points, while a larger value of C corresponds with a smaller margin with fewer interior points [50]. This problem is solved in a way similar to that of the hard margin, by first rewriting the optimisation task as a Lagrangian optimisation task, solving for the Lagrangian Multipliers and using the multipliers to find the values of \mathbf{w} and b . The above problem is still convex and can thus be written as a Lagrangian

$$\mathcal{L}(\mathbf{w}, b, \xi, \lambda, \mu) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \lambda_i \left[y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 - \xi_i \right] + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \mu_i \xi_i \quad (6.2.5)$$

(6.2.6)

where λ and μ are the Lagrange Multipliers. The KKT conditions with the additional slack variables are:

$$\frac{\delta}{\delta \mathbf{w}} \mathcal{L} = \mathbf{0} \Rightarrow \mathbf{w} = \sum_{i=0}^N \lambda_i y_i \mathbf{x}_i \quad (6.2.7)$$

$$\frac{\delta}{\delta b} \mathcal{L} = 0 \Rightarrow \sum_{i=0}^N \lambda_i y_i = 0 \quad (6.2.8)$$

$$\frac{\delta}{\delta \xi} \mathcal{L} = 0 \Rightarrow C - \mu_i - \lambda_i = 0, \quad i = [1, \dots, N] \quad (6.2.9)$$

$$\lambda [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i] = 0, \quad i = [1, \dots, N] \quad (6.2.10)$$

$$\mu_i \xi_i = 0, \quad i = [1, \dots, N] \quad (6.2.11)$$

$$\mu_i \geq 0, \quad i = [1, \dots, N] \quad (6.2.12)$$

$$\lambda_i \geq 0, \quad i = [1, \dots, N] \quad (6.2.13)$$

Substituting the above equations into the Wolfe Dual Representation in a way similar to the hard margin gives to following optimisation problem:

$$\max_{\lambda} \left(\sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{j=1}^N \sum_{i=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \right) \quad (6.2.14)$$

$$\text{subject to } \sum_{i=1}^N \lambda_i y_i = 0 \quad (6.2.15)$$

$$0 \leq \lambda_i \leq C, \quad i = [1, \dots, N] \quad (6.2.16)$$

Notice that the only difference between the above optimisation problem and the original problem (6.1.30) is the use of the scalar multiplier C to limit the upper range of the individual Lagrange multipliers. It is thus not necessary to explicitly calculate the values of ξ and μ to find the values of λ .

Once the values for λ have been calculated, they can be substituted back into the KKT equations as described for the hard margin to find the values of \mathbf{w} and b which maximise the soft margin while minimising the slack variables.

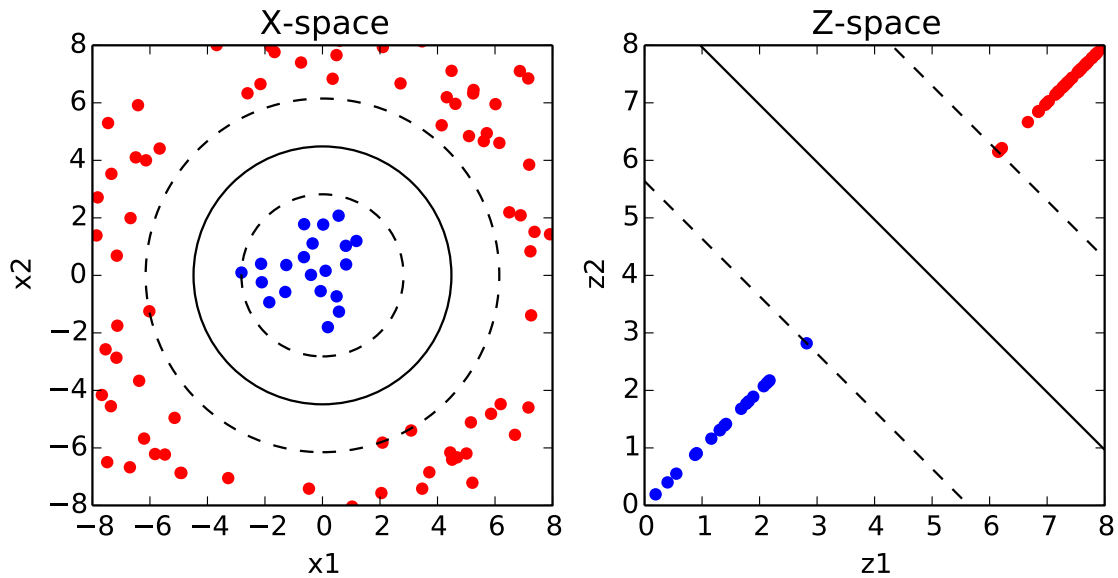


Figure 6.3.1: Example of how a transform from the \mathcal{X} -space to the \mathcal{Z} -space could make a nonlinear problem linear.

6.3 Nonlinear Classifier

SVMs are primarily designed to classify linearly separable data sets and can be extended to accurately classify interleaved data sets by using the soft margin described in the previous section. The extended SVM will still however be unable to accurately classify nonlinear data sets, such as those shown in the left image of **Figure 6.3.1**.

A simple solution to the problem of nonlinear data sets is to simply transform the data sets in such a way that they represent a linear problem. The transformed data set can then be classified using an SVM. These transformations can however increase the training and classification time of SVMs, especially when transforming the data to a feature space with a high dimension.

This section describes how a transformation affects the SVM algorithm. It also describes how kernel functions can be used to classify data sets in the new feature space without explicitly applying a transform, which can greatly increase the training and classification speed.

6.3.1 Feature Space Transform

Suppose there is a mapping function $\Phi(\mathbf{x})$ such that:

$$\Phi : \mathcal{X}^d \mapsto \mathcal{Z} \quad (6.3.1)$$

The mapping function can then be used to map any data point \mathbf{x} in the original

d -dimensional feature space \mathcal{X} to a point \mathbf{z} in the new feature space \mathcal{Z} :

$$\mathbf{z} = \Phi(\mathbf{x}) \quad (6.3.2)$$

An example of a nonlinear data set is shown in **Figure 6.3.1**. The left image represents the nonlinear data set in the \mathcal{X} feature space which contains two labelled classes. Every form of a linear classifier will always perform poorly when classifying the data set shown, making the basic SVM unusable. This specific data set can however be transformed in such a way that it is linear in the \mathcal{Z} -space, by applying the following transformation:

$$\Phi(\mathbf{x}) = \begin{bmatrix} \sqrt{x_1^2 + x_2^2} \\ \sqrt{x_1^2 + x_2^2} \end{bmatrix}$$

The data set in the \mathcal{Z} feature space can now be used to train an SVM, which will give a decision boundary and margins similar to those shown. The equivalent nonlinear decision boundary and margins in the \mathcal{X} -space are also shown.

The example above shows how a transform can be used to classify a nonlinear data set using a linear classifier. The same set of calculations as described in the previous two sections are performed on the transformed data set to find λ , \mathbf{w} and b which maximise the margin in the \mathcal{Z} -space. A new data point can be classified by simply transforming it to the \mathcal{Z} -space and using the decision boundary trained by the SVM.

There are however two problems associated with feature space transformations. The first problem is finding a transformation which transforms a nonlinear data set in the \mathcal{X} -space to a linear set in the \mathcal{Z} -space. This can be extremely difficult when the original feature space has a large number of dimensions. Secondly, using transformations to manipulate a nonlinear feature space introduces an overhead during the training and classifying stages, since all the points need to be mapped to a new feature space. The overhead can become especially large when the new feature space has a large number of dimensions, since the transformation of each point becomes more complex. The next section shows how kernel functions are used to circumvent these problems by avoiding the explicit transformation of data points.

6.3.2 Kernel Functions

Notice that the final optimisation problems (6.1.30) and (6.3.5) for the hard and soft margins contain inner products for two vectors \mathbf{x}_i and \mathbf{x}_j . Also, expanding the final

classifier (6.1.35) by substituting the first Lagrange constraint (6.1.34) gives

$$y(\mathbf{x}) = \begin{cases} 1, & \sum_{i=1}^{N_s} \lambda_i y_i [\mathbf{s}_i^T \mathbf{x}] + b \geq 0 \\ -1, & \text{otherwise} \end{cases} \quad (6.3.3)$$

which also contains an inner product indicated with the square brackets. These inner products can be leveraged to increase the training and classification speed of a nonlinear SVM, by noticing that the inner product of the data points transformed with the mapping function $\Phi(\mathbf{x})$ results in another inner product:

$$\mathbf{x}_i^T \mathbf{x} \mapsto \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x})$$

It is thus unnecessary to explicitly transform the data points to the \mathcal{Z} -space if a function $K(\mathbf{x}', \mathbf{x})$, commonly known as a kernel function, can be found that models the transformed inner product:

$$K(\mathbf{x}', \mathbf{x}) = \Phi(\mathbf{x}')^T \Phi(\mathbf{x}) \quad (6.3.4)$$

Using a Kernel Function

An example commonly used [48, 41] to illustrate the use of kernel functions is for the mapping function:

$$\Phi(\mathbf{x}) : \mathbf{x} \in \mathcal{R}^2 \mapsto \mathbf{z} = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

Calculating the inner product of two points \mathbf{x} and \mathbf{y} transformed with the above function gives:

$$\begin{aligned} \Phi(\mathbf{x})^T \Phi(\mathbf{y}) &= \begin{bmatrix} x_1^2 & \sqrt{2}x_1x_2 & x_2^2 \end{bmatrix} \begin{bmatrix} y_1^2 \\ \sqrt{2}y_1y_2 \\ y_2^2 \end{bmatrix} \\ &= x_1^2y_1^2 + 2x_1y_1x_2y_2 + x_2^2y_2^2 \\ &= (x_1y_1 + x_2y_2)^2 \\ &= (\mathbf{x}^T \mathbf{y})^2 \end{aligned}$$

The inner product of the transformed data points is thus equal to the square of the inner product of the original data points for the given transformation. The SVM could

thus be trained using the kernel function

$$K(\mathbf{x}', \mathbf{x}) = (\mathbf{x}'^T \mathbf{x})^2$$

and rewriting the soft margin optimisation problem (6.3.5) as

$$\max_{\lambda} \left(\sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{j=1}^N \sum_{i=1}^N \lambda_i \lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \right) \quad (6.3.5)$$

$$\text{subject to } \sum_{i=1}^N \lambda_i y_i = 0 \quad (6.3.6)$$

$$0 \leq \lambda_i \leq C, \quad i = [1, \dots, N] \quad (6.3.7)$$

Any data point \mathbf{x} can then be classified by substituting the kernel function in (6.3.3) and using the trained values of λ and b :

$$y(\mathbf{x}) = \begin{cases} 1, & \sum_{i=1}^N \lambda_i y_i K(\mathbf{s}_i, \mathbf{x}) + b \geq 0 \\ -1, & \text{otherwise} \end{cases} \quad (6.3.8)$$

Choosing a Kernel Function

Assume $K(\mathbf{x}', \mathbf{x})$ is a continuous symmetric function that satisfies the following condition:

$$\int_S \int_S K(\mathbf{x}', \mathbf{x}) g(\mathbf{x}') g(\mathbf{x}) d\mathbf{x}' d\mathbf{x} \geq 0 \quad (6.3.9)$$

and

$$\int_S g(\mathbf{x})^2 d\mathbf{x} < +\infty \quad (6.3.10)$$

where $\mathbf{x} \in \mathcal{R}^d$ and S is a finite subset of \mathcal{R}^d . Then according to Mercer's Theorem [48, 41] there exists a Hilbert space \mathcal{H} and a mapping function $\Phi(\mathbf{x})$ such that:

$$K(\mathbf{x}', \mathbf{x}) = \Phi(\mathbf{x}')^T \Phi(\mathbf{x})$$

Note that Mercer's Theorem does not indicate what the mapping function or the transformed space is, only that they exist for the given kernel function. Three types of kernels typically used in pattern recognition problems which satisfy Mercer's conditions are Polynomials, Radial Basis/Gaussian and Hyperbolic Tangent kernels [41].

Their respective kernel functions are:

$$K_{poly}(\mathbf{x}', \mathbf{x}) = (\mathbf{x}'^T \mathbf{x} + 1)^q, \quad q > 0 \quad (6.3.11)$$

$$K_{gauss}(\mathbf{x}', \mathbf{x}) = \exp \left(-\frac{\|\mathbf{x}' - \mathbf{x}\|^2}{\sigma^2} \right) \quad (6.3.12)$$

$$K_{hyper}(\mathbf{x}', \mathbf{x}) = \tanh \left(\beta \mathbf{x}'^T \mathbf{x} + \gamma \right) \quad (6.3.13)$$

The choice of kernel function to use is dependent on the data. The libSVM authors suggest in [49] to train a model using a Gaussian kernel first, which can then be used as a baseline for comparing the performance of other kernels if necessary. The choice of a Gaussian kernel is motivated by the low complexity of the classifier and the higher numerical stability when compared to the other kernels.

6.4 Multiclass Classifier

The final modification to the basic two-class linearly separable SVM classifier which is of importance to this project is the addition of a framework to classify multiple classes. This extension is needed to be able to differentiate between the 17 hand poses our system needs to recognise.

Three classification techniques are briefly discussed by [41] for extending the SVM for multiclass problems, namely One-against-all (one-all), One-against-one (one-one) and Error Correction Coding. Both one-all and one-one, along with a fourth technique called Direct Acyclic Graph SVM (DAGSVM), were evaluated in [53].

This section discusses only the one-one and one-all classification techniques, since they are the most common techniques and are easy to implement. Furthermore, the performance of the one-one and one-all classification techniques are similar to other techniques according to [53].

6.4.1 One-against-all

The one-all algorithm divides an M-class data set into M discriminant problems where each problem consist of positive and negative examples of one of the M classes. One SVM is trained for each of the M classes, such that the positive examples lie on the positive side of the decision boundary as determined by the orientation of \mathbf{w} . The individual training problems along with possible decision boundaries are shown in the top row of **Figure 6.4.1**.

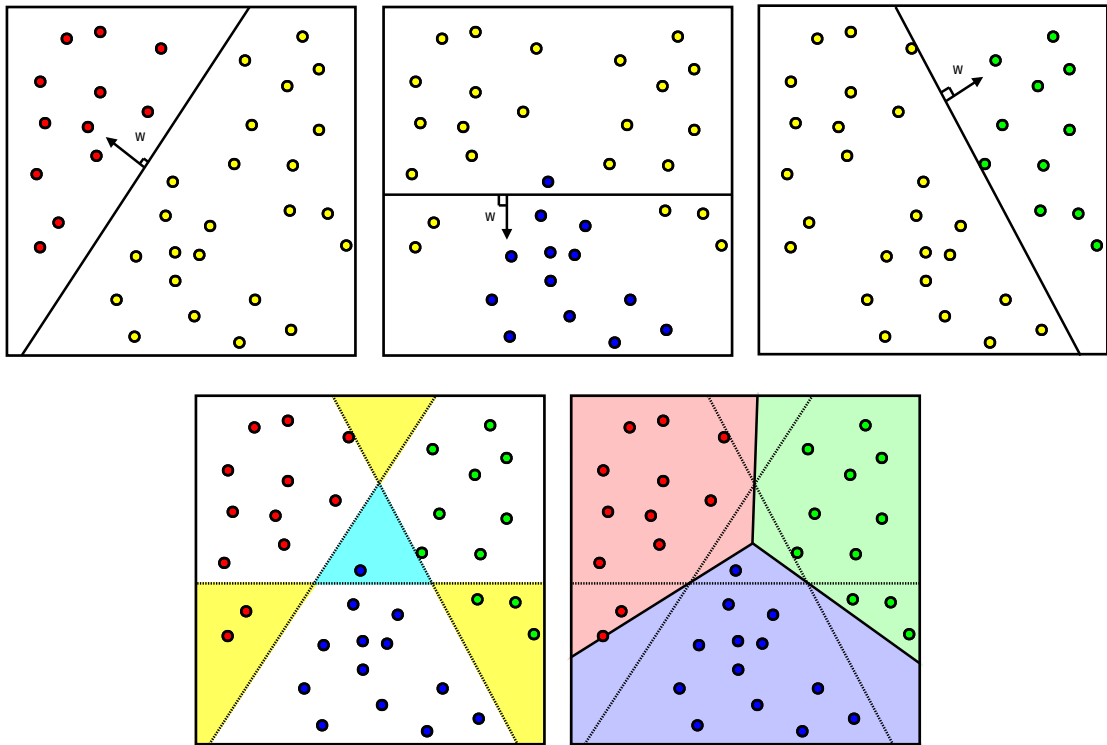


Figure 6.4.1: Example of a One-against-all multiclass classifier. The top row shows 3 potential classifiers trained from the three 2-class problems, including the direction of the \mathbf{w} vector. The yellow data points represent the negative examples for each of the individual classifiers. The bottom row shows the combined classifier along with the indeterminate regions and the final classification regions.

A new point can be classified by testing it against all M classifiers and assigning it to the class corresponding to the classifier which has the highest response value. The combined classifier can be written as

$$y(\mathbf{x}) = \arg \max_i g_i(\mathbf{x}) \quad i = [1, \dots, N] \quad (6.4.1)$$

where $g_i(\mathbf{x})$ is the discriminant function corresponding to class i .

The one-against-all algorithm suffers from imbalanced training sets for the individual classifiers, since there are generally more negative than positive training examples. These imbalanced training sets can produce classifiers that perform badly compared to classifiers trained using balanced sets.

Another problem is the occurrence of indeterminate regions, shown in **Figure 6.4.1** on the bottom row, where more than one classifier have a positive response (yellow regions) or no classifier has a positive response (cyan region). In these regions, (6.4.1) will simply assign the class corresponding to the decision boundary “nearest” to the point. Note that as mentioned in **Section 6.1**, SVM classifiers do not use a Euclidean

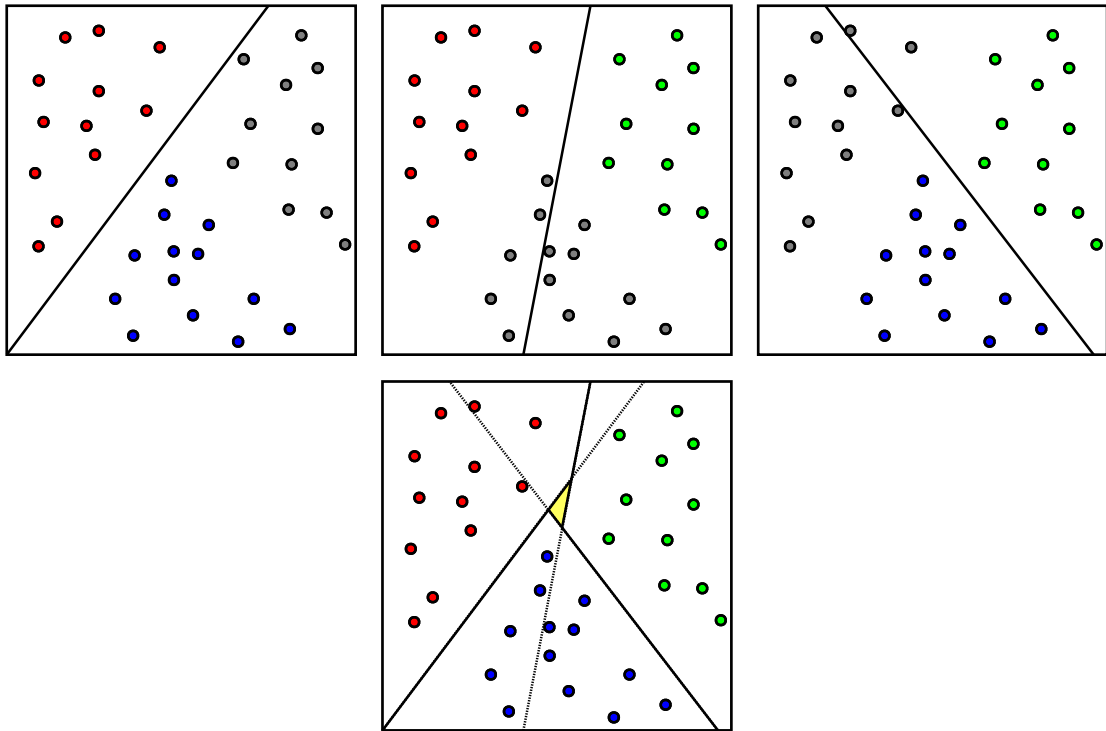


Figure 6.4.2: Example of a One-against-one multiclass classifier. The top row shows 3 potential classifiers trained from the three 2-class problems. The grey data points indicate the data which are not considered during the training of an individual classifier. The bottom image shows the decision boundaries of the combined classifier. The yellow area indicates an indeterminate area, where all classes receive an equal number of votes when classifying a new data point inside the region.

distance to measure the distance between a point and a boundary, thus the distance will be measured differently for each of the individual classifiers. The second image in the bottom row of **Figure 6.4.1** shows a rough estimate of the classification regions of the example classifier.

6.4.2 One-against-one

One-against-one similarly to one-against-all divides an M -class problem into several 2-class problems for training. The algorithm divides an M -class problem into $\frac{M(M-1)}{2}$ 2-class problems, where each problem only considers the data of two classes, ignoring the other classes. A separate SVM classifier is trained for each problem and then used in a combined classifier when classifying a data point. An example of this training technique is shown in **Figure 6.4.2** for a 3-class problem.

The paper of [53] advised using a voting system for classifying a new data point, where the class with the most votes is assigned to the data point. The voting algorithm is shown in **Algorithm 2**. For the case when more than one class have the most votes, [53] simply used the class with the lowest index, though admits that a better strategy can be used.

Algorithm 2 One-against-one classification algorithm.

```

for  $i \in [1 \dots M]$  do
  for  $j > i \&\& j \in [1 \dots M]$  do
    if  $g_{ij}(\mathbf{x}) \geq 0$  then
      Assign vote to class (i or j) corresponding the positive  $g_{ij}(\mathbf{x})$ .
    else
      Assign vote to class (i or j) corresponding the negative  $g_{ij}(\mathbf{x})$ .
  return Class with most votes.

```

One disadvantage of the one-against-one compared to the one-against-all algorithm is that a larger number of classifiers is trained and used for classification. The comparison between different multiclass SVMs of [53] did however experimentally show that these classifiers can in some cases give faster training and classifying times. The training times are likely faster because fewer points are considered during training of an individual one-against-one classifier when compared to a individual one-against-all classifier.

6.5 Summary

In this chapter we discussed Support Vector Machines, which are used as the basis of our Pose Classifiers. **Section 6.1** documented the theory of the basic two-class linear SVM classifier using a hard error margin for training. **Section 6.2** described how the basic theory changes when using a soft error margin to train SVM classifiers. The usage of kernel function to create nonlinear SVM classifiers were described in **Section 6.3**. The chapter ended with **Section 6.4** which discussed multiclass SVM classifiers.

Chapter 7

Pose Classification

The final step in the classification process is to use the estimated joint positions to classify the pose of the hand. This chapter will describe the techniques used to train and classify the different hand poses from the joint estimates discussed in **Chapter 5**. The chapter is divided into two parts. The first part discusses the features extracted from the joint estimates. The second part will discuss how these joints can be used to train a pose model using the SVMs described in **Chapter 6**.

7.1 Joint Feature Extraction

The extraction of accurate and robust features is critical to the performance of the final pose classifier. The joint features extracted must be able to describe the pose independent of the hand's rotation and scale. Furthermore, they should be fast and have a negligible effect on the speed of the final classifier. The classifier of [7] used the 2D joint coordinates directly for pose classification. This system showed promising results, but it was decided to investigate other features to see if the performance could be increased further.

This section will discuss the three different types of features investigated, namely Position, Transformed and Angle Features. Each feature type is further divided into a 2D and a 3D case, giving a total of six unique feature sets.

7.1.1 Position Features

Position features are simply a normalisation and repackaging of the 21 joint coordinates retrieved from the joint estimator. These features are fast to extract and simple to implement. They also provide a convenient baseline to which the other feature types

can be compared.

The 2D case uses the 2D image coordinates retrieved from the joint estimator. These coordinates are represented as a 42D feature vector

$$\mathbf{f}_2 = \begin{bmatrix} x_0, & y_0, & \dots, & x_{20}, & y_{20} \end{bmatrix} \quad (7.1.1)$$

where x_i and y_i denote the image coordinates of joint i . The coordinates are then normalised using the width and the height of the image such that the values fall in the range $[-1.0, 1.0]$.

The 3D case uses a similar 63D feature vector, which also includes the depth values z_i of the various joint coordinates:

$$\mathbf{f}_3 = \begin{bmatrix} x_0, & y_0, & z_0, & \dots, & x_{20}, & y_{20}, & z_{20} \end{bmatrix} \quad (7.1.2)$$

The depth values are normalised to the range $[0, 1.0]$ using the minimum and maximum expected depth values (0.3m and 1.45m respectively).

Joints which are not found during the joint estimation phase are simply placed at zero coordinates located at the top left of the image. The coordinates of these joints are $(0, 0)$ and $(0, 0, 0)$ for the 2D and 3D cases respectively.

This feature type is not invariant to translation, rotation or scaling of the coordinates, which can lead to incorrect classification of small pose variations. This problem can be partially solved by ensuring sufficient pose variations are present in the training set. The next two feature types attempt to solve this problem using different approaches.

7.1.2 Transformed Features

Transformed Features make use of the joint coordinates in a way similar to Position Features. The pose variation problem is addressed by applying a transformation to these coordinates, which consists of three smaller transformations: a translation, a rotation and a scaling transformation. These transformations try to manipulate the joint coordinates in such a way that the final position, orientation and scale are the same for all pose variations. This constrains the pose classifier to model only the hand pose.

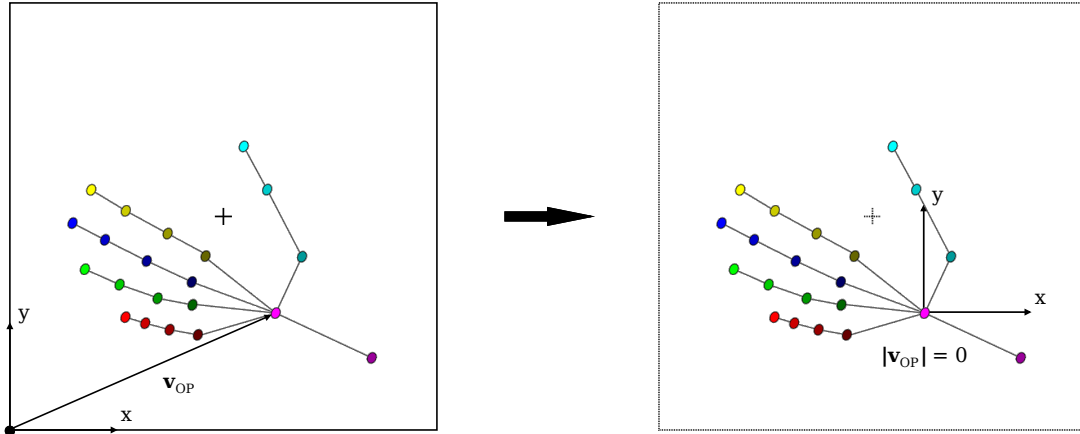


Figure 7.1.1: A simple translation transformation.

Translation

The first transformation attempts to make the feature vectors invariant to all translations. The features can be made invariant to translations by moving the static image origin to a new dynamic origin. The pose classifier will be invariant to translations if the new dynamic origin remains stationary relative to the hand joints for all translations.

Figure 7.1.1 shows how the translation vector is calculated. The coordinates of the palm joint is chosen as the new dynamic origin. This choice is motivated by the high recognition rate for the palm region, making estimation of the palm joint more reliable. The static origin is located at $(0,0)$ for the 2D case and at $(0,0,0)$ for the 3D case. The translation vector \mathbf{t} is calculated as the negative of \mathbf{v}_{OP} , the vector connecting the palm and wrist joints. The matrices used to apply the translation for the 2D and 3D cases are

$$\mathbf{T}_2 = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad (7.1.3)$$

and

$$\mathbf{T}_3 = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.1.4)$$

where t_x , t_y and t_z are the x-, y- and z-components of \mathbf{t} .

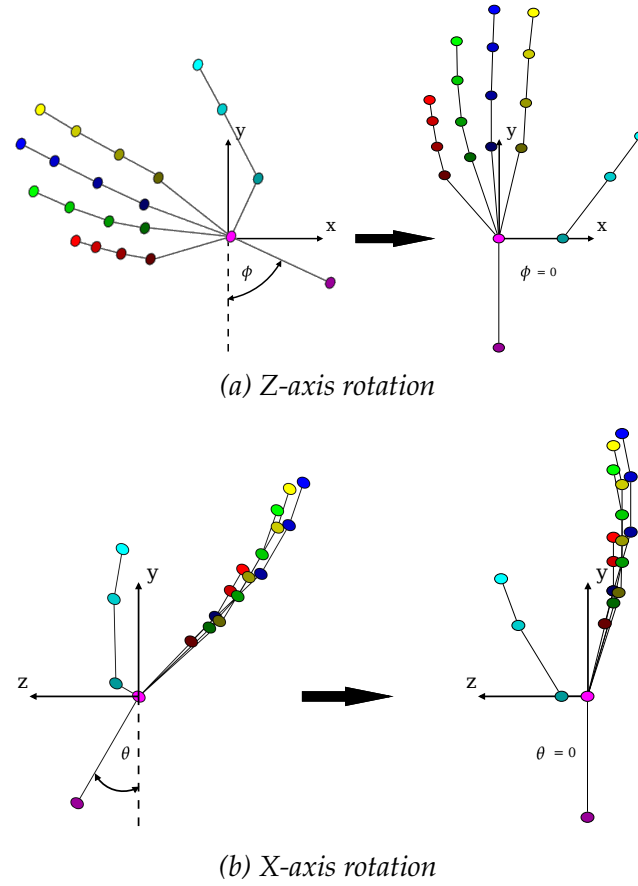


Figure 7.1.2: The rotation transformations.

Rotation

The second transformation applies a rotation to all the joints. This transformation rotates the hand joints in an upright position as shown in **Figure 7.1.2**. The origin of rotation is again chosen as the palm joint for the same reasons as previously described. The angle of rotation ϕ is calculated as the smallest angle between the vector connecting the palm and wrist joints and the y-axis. The following matrices are used to apply the described rotation for the 2D and 3D cases:

$$\mathbf{R}_2 = \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (7.1.5)$$

and

$$\begin{aligned} \mathbf{R}_3 &= \mathbf{R}_x \mathbf{R}_z \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \quad (7.1.6)$$

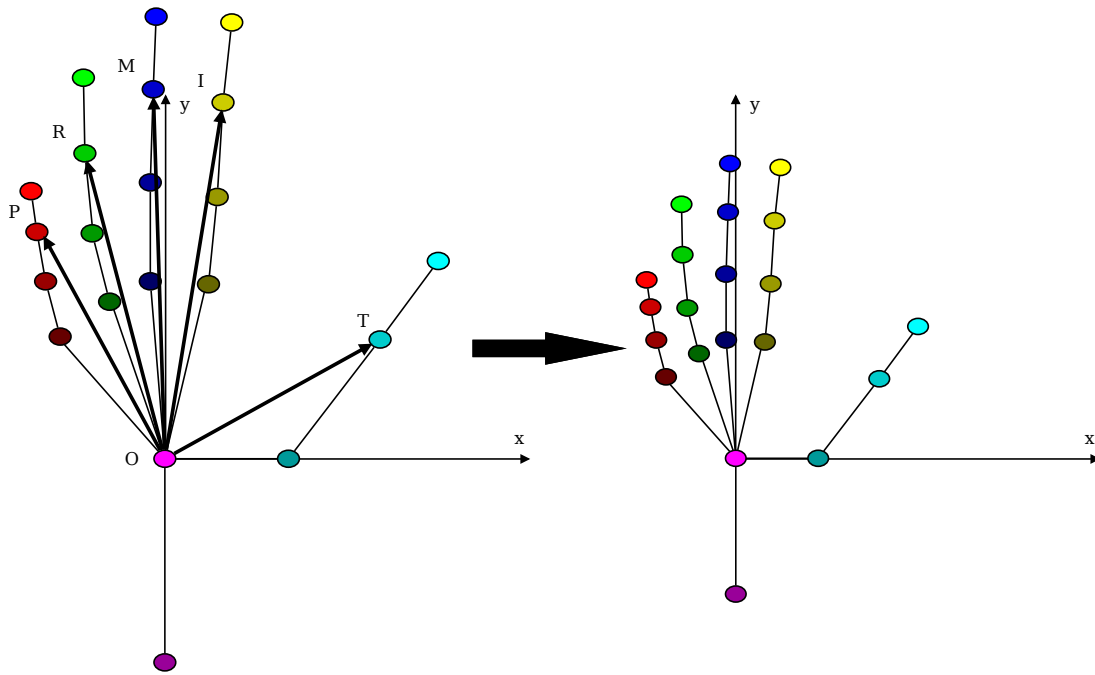


Figure 7.1.3: The scaling transformation. *P, R, M, I and T indicate the second joints of the five fingers and O indicates the palm joint.*

where \mathbf{R}_2 and \mathbf{R}_3 are the rotation matrices for the 2D and 3D cases respectively.

One potential disadvantage of this transformation is its dependence on accurate estimates of the palm and wrist joints. The Region Classifier test results of **Chapter 4** do however indicate a high recognition rate for the palm and wrist regions, which leads to accurate joint estimations from the Joint Estimator when using synthetic images.

Real world images are more problematic because the cropping of the wrist region differs to that of the synthetic images. This leads to a wrist joint which is either closer to the palm or further up the arm than expected from the synthetic training images. The images do however show that the vector connecting the palm and wrist joints still gives a good indication of the hand's orientation.

Scaling

The final transformation is a scaling transformation. This transformation addresses the scale variations which occur because of variations in physical hand sizes and distances between the hand and the depth sensor. The transformation achieves this invariance by scaling the palm area of the hand such that all hands have a similar palm size. Furthermore, the transformation normalises the joint coordinates to the range $[-1, 1]$ to help prevent numerical errors during pose classification.

The scale factor used is calculated using the palm joint and the second finger joint of each finger, as illustrated in **Figure 7.1.3**. The second finger joints were chosen because they are considered the most stable, as shown in results of **Chapter 5**. The distance values of the second finger joints do change when the fingers bend, but as shown by the results of **Chapter 5** they are still the most stable joints. We could have used the knuckle joints, which will have a minimal change in depth when the fingers bend, but these joints would then suffer from occlusion, making them even more unstable. The scale factor is calculated as:

$$s^{-1} = 2/5 (|V_{OP}| + |V_{OR}| + |V_{OM}| + |V_{OI}| + |V_{OT}|) \quad (7.1.7)$$

The factor $2/5$ normalises the sum of the distances to calculate the average, then halves the scaling factor s to ensure the coordinates are in the range $[-1, 1]$. The matrices used to scale the 2D and 3D cases respectively are

$$\mathbf{S}_2 = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (7.1.8)$$

and

$$\mathbf{S}_3 = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.1.9)$$

Joint Transformation

The above transformations can be combined into one matrix:

$$\mathbf{A}_d = \mathbf{S}_d \mathbf{R}_d \mathbf{T}_d \quad (7.1.10)$$

where $d \in \{2, 3\}$ for the 2D and 3D cases respectively. This transformation is similar to the similarity transformation, except for the order in which the individual transformations are applied. The homogeneous joint coordinates are represented as a matrix on which the transformation is applied:

$$\mathbf{X}_2 = \begin{bmatrix} x_0 & x_1 & \dots & x_{20} \\ y_0 & y_1 & \dots & y_{20} \\ 1 & 1 & \dots & 1 \end{bmatrix} \quad (7.1.11)$$

and

$$\mathbf{X}_3 = \begin{bmatrix} x_0 & x_1 & \dots & x_{20} \\ y_0 & y_1 & \dots & y_{20} \\ z_0 & z_1 & \dots & z_{20} \\ 1 & 1 & \dots & 1 \end{bmatrix} \quad (7.1.12)$$

The final transformation is performed using matrix multiplication:

$$\mathbf{X}'_d = \mathbf{A}_d \mathbf{X}_d \quad (7.1.13)$$

Dimension Reduction

The coordinates are represented by a feature vector similar to the one shown in **Section 7.1.1**. The final step is to reduce the dimension of this feature vector. The 2D and 3D transformed feature vectors \mathbf{f}'_2 and \mathbf{f}'_3 are in the form

$$\mathbf{f}'_2 = \left[x'_{0'}, y'_{0'} \dots, 0, 0, \sim 0, y'_{20} \right] \quad (7.1.14)$$

and

$$\mathbf{f}'_3 = \left[x'_{0'}, y'_{0'}, z'_{0'} \dots, 0, 0, 0, \sim 0, \sim 0, z'_{20} \right] \quad (7.1.15)$$

where x'_i , y'_i and z'_i indicate the transformed coordinates. The values indicated by ~ 0 are near zero values, which occur because of precision errors when applying the rotation matrix. These values can essentially be rounded down to zero.

The feature vectors above show that the palm joint will always be placed at the origin because of the translation transformation. Furthermore, the rotation transformation will force the wrist joint onto either the y - or z -axis. The dimensions of the wrist and palm joints with fixed values (~ 0 or 0) can thus be removed from the feature vector without affecting the final pose classification.

The final dimension assigned to the wrist vector (either y'_{20} or z'_{20}) is also removed. The removal is motivated by the fact that the wrist region is subject to a large amount of variation when using real data, because of the wrist cropping problems previously discussed in this section. The final 38D and 57D feature vectors are of the form

$$\mathbf{f}'_2 = \left[x'_{0'}, y'_{0'} \dots, x'_{18'}, y'_{18'} \right] \quad (7.1.16)$$

and

$$\mathbf{f}'_3 = \left[x'_0, y'_0, z'_0, \dots, x'_{18}, y'_{18}, z'_{18} \right] \quad (7.1.17)$$

7.1.3 Joint Angles

The last set of features uses the angles between various joints for training and classifying hand poses. The motivation behind using joint angles instead of directly using the joint coordinates is two-fold. Firstly, the angle between two vectors is invariant to the scale of the two vectors, making joint angles more resistant to variations in scale. Secondly, the angles between joints are less affected by the shape of a hand than the coordinates of the joints. This reasoning does however assume that the joint coordinates are accurately estimated.

Calculating the Joint Angles

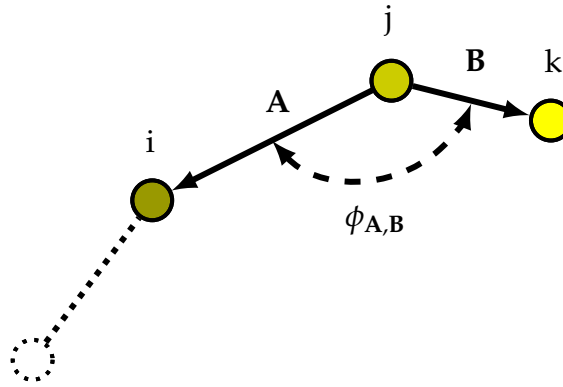


Figure 7.1.4: The angle $\phi_{A,B}$ between three joints i , j and k .

The angle between three joints i , j and k as shown in **Figure 7.1.4** is calculated using the dot product

$$\phi_{A,B} = \arccos \left(\frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} \right) \quad (7.1.18)$$

where \mathbf{A} and \mathbf{B} are the vectors from joint j to joints i and k respectively.

The implemented system considers the joint angles between the joints of an individual finger and the palm. The wrist joint is not used for the same reasons discussed in the previous section. This system only derives meaning from how much each finger is bent relative to the palm, thus focusing on the shape of the hand and potentially providing a more robust feature set. An illustration of some of the joints that are used

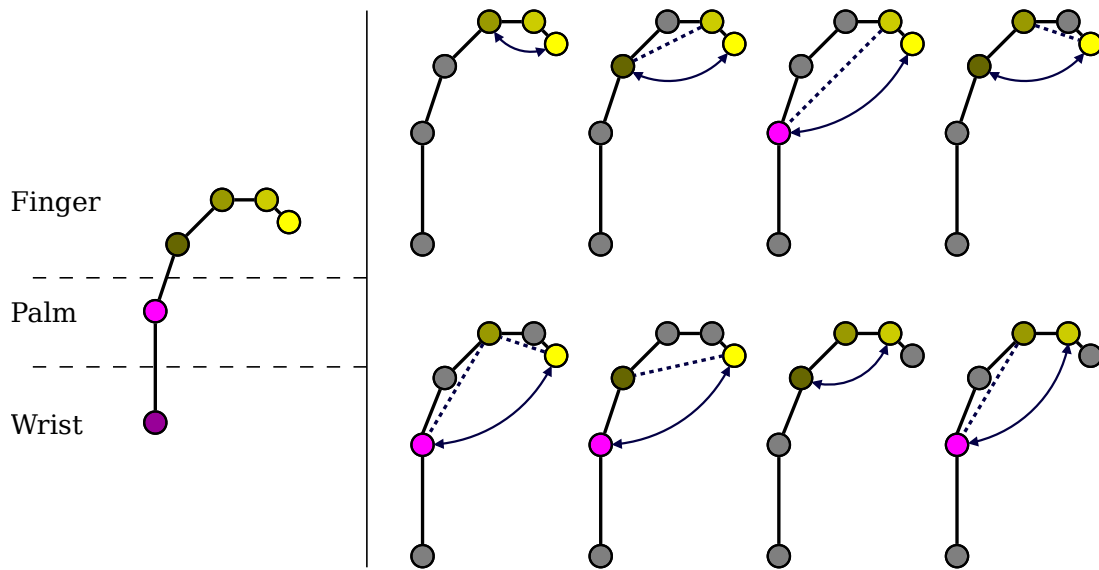


Figure 7.1.5: A view of the joints of a bent index finger, the palm and wrist from the side, along with a few example angles used in the feature set.

in the feature set is shown in **Figure 7.1.5**.

The same measurements are used for the 2D and 3D case of these features. The 2D case uses the 2D image coordinates directly to calculate the angles, while the 3D case uses the 3D coordinates which include the depth values.

It can be argued that calculating angles from 2D images is risky, since they are measure in a projective space. They are however still present, used as an indication of the orientation of vectors in relation to each other. The angles do not however give a definitive indication of how much a finger is bent, but will give an indication of what is perceived from the camera. In retrospect, we believe that a dot product between the measured vectors might have been a better measurement, since it is well defined in all dimensions and gives an indication of the orientation of two vectors in relation to each other, similar to angles. Further research could investigate the use of dot products between various vectors as a potentially improvement on the angle features.

Creating the Feature Vector

A total of 10 angles can be measured per finger using the angle measurement configuration described above, with the exception of the thumb for which 4 angles can be measured. This gives a total of 44 possible measurements. Each measurement is stored in a 44D feature vector, starting with the measurements from the little finger and continuing up to the thumb measurements. The final feature vector is in the form

$$\mathbf{f}_d = \begin{bmatrix} a_0, & a_1, & \dots, & a_{42}, & a_{43} \end{bmatrix} \quad (7.1.19)$$

where a_i is the i^{th} angle measured and $d \in \{2, 3\}$ for the 2D and 3D cases respectively.

7.2 Pose Classifier

The Pose Classifier is the final component in the pose recognition system. It receives a set of joint features extracted from a depth image as described in the previous section and classifies them using a pose model. The pose models of this project are implemented using Support Vector Machines as described in the previous chapter. The training and classification of the SVM models are implemented using the libSVM [54] library. The next two sections will give a description of the SVM model used and the training process.

7.2.1 Model Description

The pose classification problem requires a multiclass classifier to distinguish between the 17 different poses. Furthermore, the training set of joint features may not be linearly separable because of sensor noise and various hand rotations. The non-separable nature of the set is further enhanced by the inclusion of pose variations such as the “open” and “closed” hand poses. This problem motivated the use of a multiclass, kernel-based SVM classifier using a soft margin.

The one-against-one algorithm is used by libSVM to train multiclass classifiers according to the FAQ [54] and [53]. The creators of libSVM chose this algorithm over one-against-all, because it gives similar results while also having a shorter training time.

A nonlinear classifier is modelled using a Gaussian kernel, which the guide of [49] suggested when initially designing an SVM model for a particular data set. We found that the kernel performed well and used it for all the SVM models trained, though further testing of other kernels is needed to determine if it is the best choice. The form of the libSVM Gaussian kernel is

$$K(\mathbf{x}', \mathbf{x}) = \exp\left(-\gamma \|\mathbf{x}' - \mathbf{x}\|^2\right), \quad \gamma > 0$$

The form of the nonlinear classifier and the error margin can be adjusted by changing the values of γ and C . The next section discusses the training of the SVM model and how the values of γ and C are chosen to maximise the performance of the final model.

7.2.2 Model Training

The pose models used in this project were trained using a subset of 20000 images from the synthetic training set used to train the Region Classifier. The subset contains examples of all the predefined poses, with various rotations and scalings applied to the 3D model. A pose trainer was built using the libSVM library, which takes as input a joint feature set and produces an SVM model.

The first step in the training process involves the extraction and packaging of the joint features. The depth images are first processed using the Region Classifier to extract the region map of each image, which is then processed by the Joint Estimator to produce the estimated joints. The joint features are extracted from the output of the Joint Estimator and collected in a joint feature set. The feature set is then processed into the format expected by libSVM.

Parameter optimisation is the second step in the training process, in which the optimal values of γ and C are found in order to prevent overfitting. A simple grid search is used for parameter optimisation as suggested by [49]. An exponentially increasing range of values is chosen for both γ and C , in our case $\left[2^{-5}, 2^{-4}, \dots, 2^1, \right]$ and $\left[2^{-1}, 2^0, \dots, 2^7, \right]$ respectively. Each combination of γ and C is used to perform 10-fold cross validation on the training set. The parameters which leads to the highest cross validation score is the optimal parameters. The final step in the training process is to use the optimal training parameters to train a model using the complete training set.

Various models are trained for testing purposes. A separate model is trained for each joint estimator and feature combination for testing purposes and to maximise the performance of each classifier. The model used during the classification stage should thus correspond to the joint estimator and feature extractor used for the best results.

7.3 Testing

This section describes the tests performed to determine the accuracy of the different pose classifiers. These tests effectively test the system as a whole using synthetically generated images. Each pose classifier consisted of a different combination of Region Classifier, Joint Estimator and Feature Extractor. Both accuracy and speed tests were performed on the various pose classifiers, as detailed below.

7.3.1 Experimental Setup

The features used for testing are extracted from the depth images generated for Testing Sets A and B. Each individual pose classifier consists of a Region Classifier, a Joint Estimator and a SVM model that uses one of the joint features described in the previous section for pose classification.

The first set of tests determined the accuracy of various classifiers. Region Classifiers RDF16_AT, RDF16_AL, RDF16_BT and RDF16_BL (described in **Chapter 4**) are used to classify the hand regions, while the joint sets are extracted using the three Joint Estimators described in **Chapter 5**. The various features described at the start of this chapter are extracted from the joints sets from which the pose is classified using a trained SVM model corresponding to the extracted feature.

The second set of tests determined the classification speed of the complete system. Region Classifiers RDF8_BL and RDF16_BL were used, since the 16-tree region classifiers trained using Training Set B had the highest accuracy, while the 8-tree variation was still able to classify depth images in real-time according to the test results of **Chapter 4**.

7.3.2 Pose Classifier Accuracy

The first set of tests determined the accuracy of the various pose classifiers when classifying Testing Set A. The results of these tests are shown in **Figure 7.3.1**. The Region Classifiers trained using the Large training subset always outperform the Smallest training subset Region Classifiers.

The tests showed that the pose classifiers using region classifiers trained using the Smallest subset of Training Set B performed better than their Training Set A counterpart in every case, while using the Larger subsets the pose classifiers using Training Set A performed better for every case. The Mean-Shift- and Reservation-based pose classifiers generally performed similarly, though showed a large improvement over the Centre of Gravity estimator when using the Position and Transform joint features.

The Position and Transform features generally perform the same, while the accuracy of the Angle features are considerably lower. The tests also show that there is little to no difference between using the 2D or the 3D joint feature variations. The best performing classifier with an accuracy of 98.71% consisted of the RDF16_AL region classifier and the Reservation Joint extractor using Transform 3D features.

A similar set of tests was performed on Testing Set B, with the results shown in **Figure 7.3.2**. These tests showed that the more general region classifier trained using Training Set B performed considerably better than the specialised classifiers from Training Set A for every case. The different joint estimators and joint features showed similar characteristics as seen when testing Testing Set A. The best performing classifier with an accuracy of 95.61% consisted of the RDF16_BL region classifier and the Reservation Joint extractor using Transform 3D features. The numerical results of the other tests are shown in **Appendix B**.

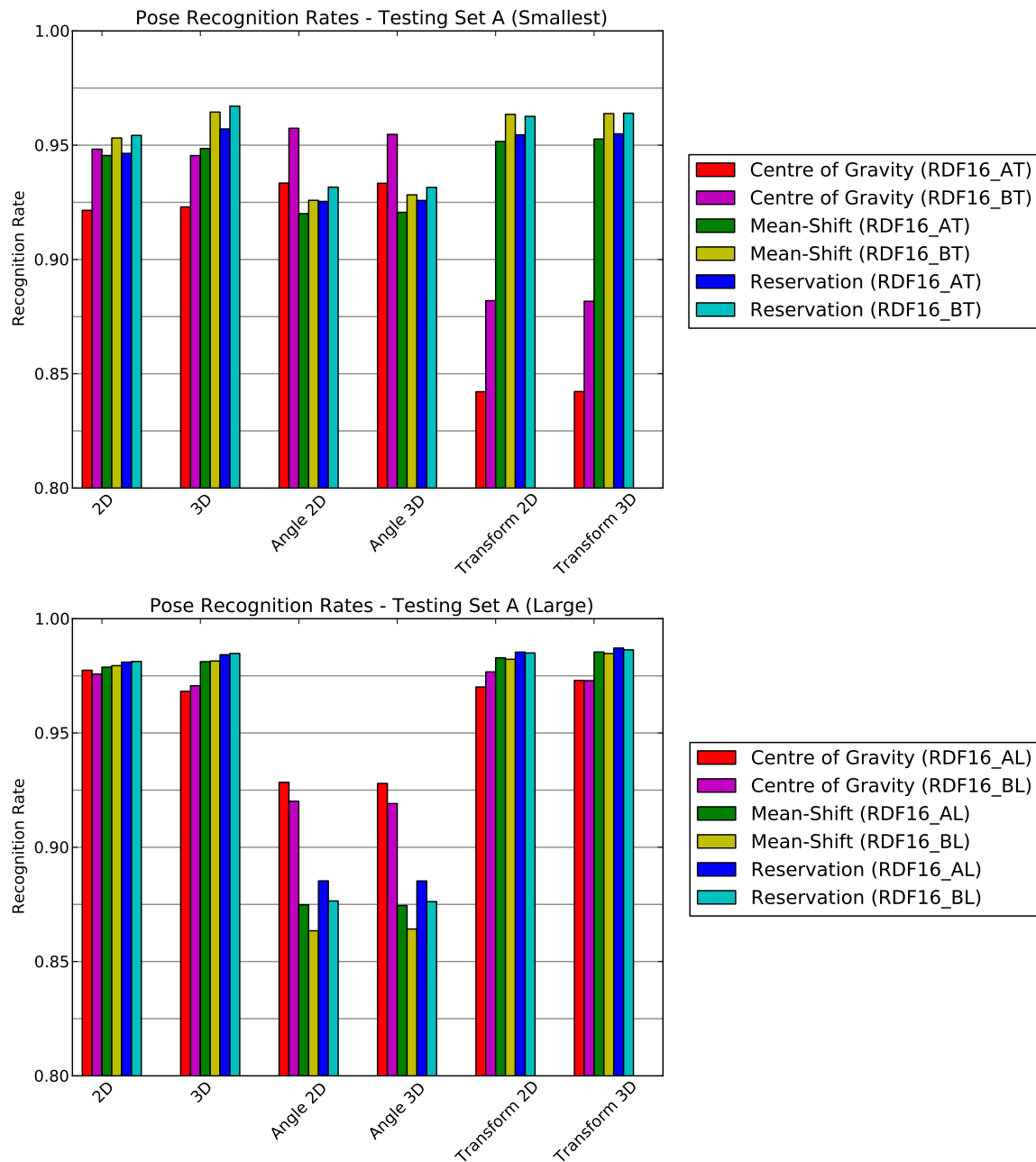


Figure 7.3.1: The results of testing the various pose classifiers on Testing Set A. The two graphs show the results when using Region Classifiers RDF16_AT (top) and RDF16_AL (bottom).

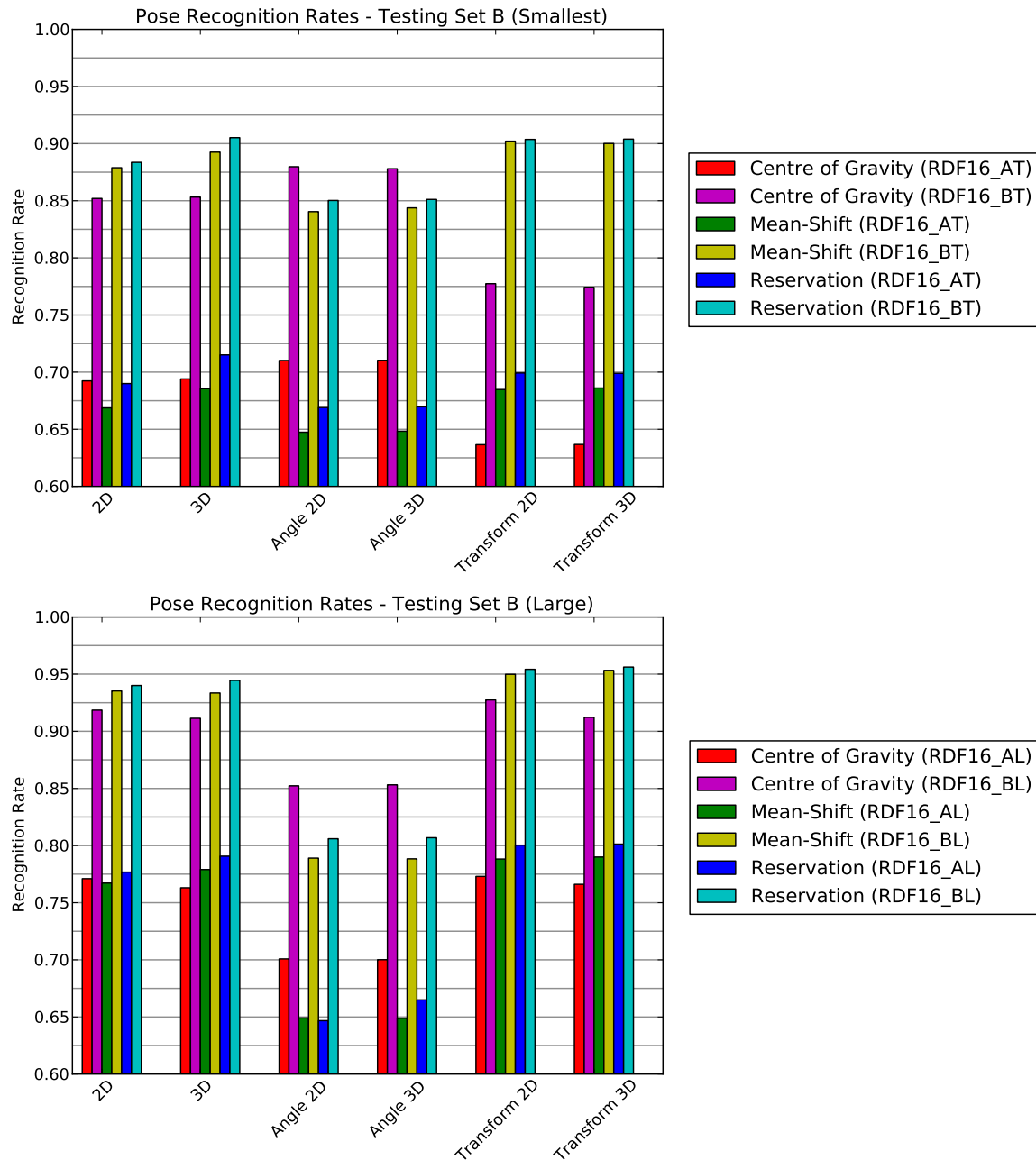


Figure 7.3.2: The results of testing the various pose classifiers on Testing Set B. The two graphs show the results when using Region Classifiers RDF16_AT (top) and RDF16_AL (bottom).

7.3.3 Confusion Properties

Confusion matrices for the results of the best pose classifier, RDT16_BL with Transform 3D joint features extracted from the Reservation joint estimates, were generated from the results similar to those of the Region Classifier in **Chapter 4**. The graphs' axes are labelled with the names of the various tested American Sign Language poses, where (xC) denotes a "closed" variant of pose x. The top graph shows the results when using Testing Set A, while the bottom graph shows the results of Testing Set B. Colour graphs of these matrices are shown in **Figure 7.3.3**, with the corresponding confusion matrices

shown in Appendix B.

The graphs show a clear diagonal line, expected given the high recognition rate of the pose classifier. The most confusion was caused by the “open” and “closed” variants of poses, which is expected given the close relationship between these variances. The confusion is however very limited, which shows that the system can differentiate between subtle differences in poses if the quality of the classification from the region classifier is high enough.

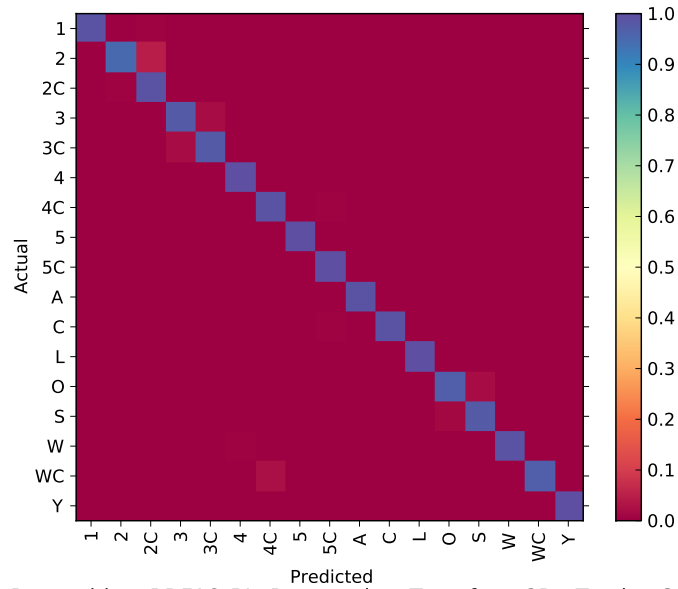
The classifier occasionally confused similar poses, with the worst confusion taking place between the 4C and WC poses. This might be due to ambiguities caused by various views of the hand, where some of the fingers become occluded.

7.3.4 McNemar Tests

We also applied McNemar’s [55] test to all of the above tests in order to compare the different results and determine which differences in accuracy are statistically significant. McNemar’s test is used to measure the probability that the difference between two results is *not* significant, by comparing the classification results of each individual data point. We can assume that the difference in accuracy of two classifiers is statistically significant if the probability is close to zero, otherwise further testing is needed to be able to determine the comparative performance of two classifiers. The results of the various McNemar tests are shown in **Figure 7.3.4**. Empty entries indicate values of zero, while entries with values smaller than 0.001 are rounded to zero and indicated as ~ 0 .

The figure shows that there is no significant difference between some of the 2D and 3D feature classifiers. This is expected, not only because of the similar results, but since the depth resolution of both the Kinect and synthetic depth images is low, which means that in most cases the extra depth dimension of the 3D classifiers does not provide more information than the corresponding 2D variation. It can also be seen that further testing is required to determine whether there is a measurable difference between the results obtained when using the Mean-Shift joint estimator as opposed to the Reservation joint estimator when using the Position joint features.

Pose Recognition: RDF16_BL, Reservation, Transform 3D - Testing Set A



Pose Recognition: RDF16_BL, Reservation, Transform 3D - Testing Set B

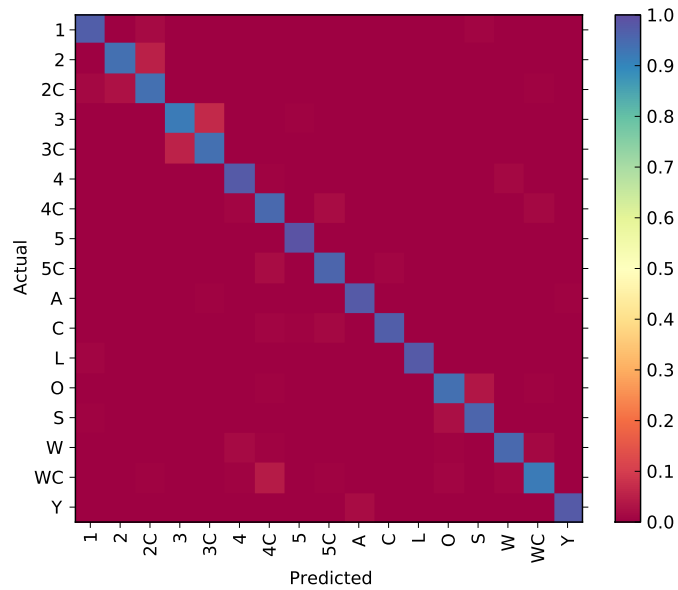


Figure 7.3.3: Confusion colour graphs indicating which poses the system confuses with which. The two graphs show the results when using the best performing pose classifier (RDF16_BL, Reservation, Transform3D) on Testing Set A (top) and Testing Set B (bottom). **Table B.3.6** and **Table B.3.7** show the corresponding confusion matrices.

	CP3D	CA2D	CA3D	CT2D	CT3D	MP2D	MP3D	MA2D	MA3D	MT2D	MT3D	RP2D	RP3D	RA2D	RA3D	RT2D	RT3D	
CP2D																		CP2D
CP3D					.057													CP3D
CA2D			.04															CA2D
CA3D																		CA3D
CT2D																		CT2D
CT3D																		CT3D
MP2D							.759					~0						MP2D
MP3D												~0						MP3D
MA2D								.199										MA2D
MA3D																		MA3D
MT2D																		MT2D
MT3D																.062	~0	MT3D
RP2D																		RP2D
RP3D																		RP3D
RA2D															.093			RA2D
RA3D																		RA3D
RT2D																	~0	RT2D

Figure 7.3.4: McNemar tests to evaluate the comparisons of **Figure 7.3.2** (bottom).

7.3.5 Pose Classifier Speed

The second set of tests determines the speed of the pose classifiers. A subset of 1632 images from Testing Set B was used to determine the average FPS of the tested Pose Classifiers. The images were classified using the complete system and the total classification time was recorded. **Figure 7.3.5** shows the results of these accuracy tests.

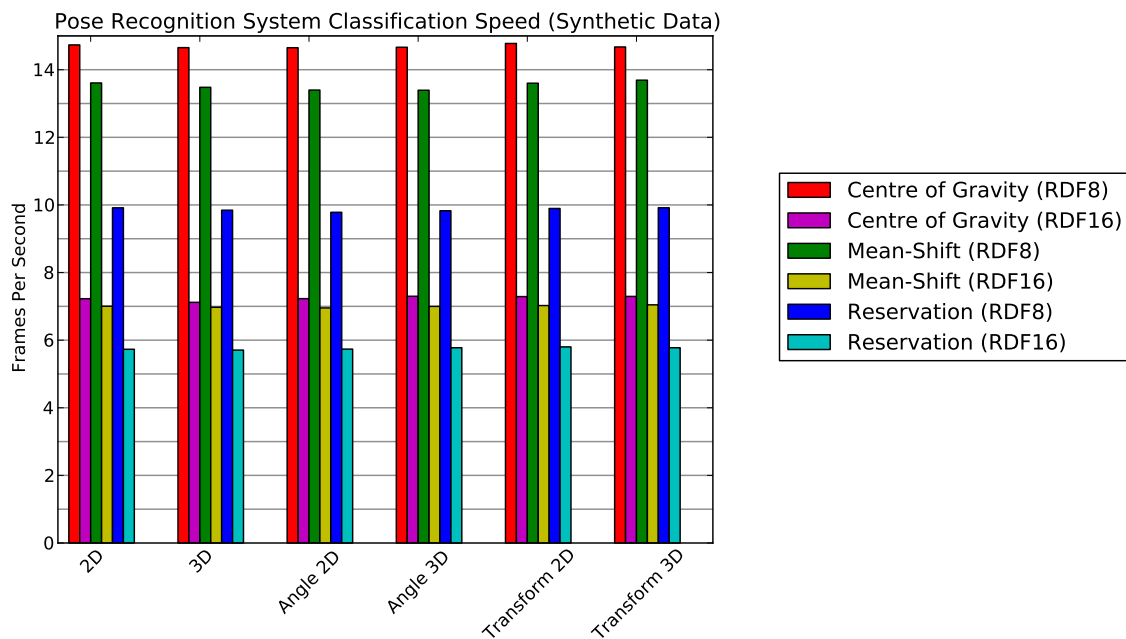


Figure 7.3.5: The results of speed tests performed on the system using the synthetic data from Testing Set B.

The results show that the Region Classifier has the largest effect on classification speed, followed by the Joint Estimator used. There is little to no difference in classification speed between using the various joint features or their SVM models. None of the classifiers tested was able to perform in real-time, which is less than ideal. Enhancing the performance of the Region Classifiers and Joint Estimators using GPU acceleration would alleviate the problem considerably, but is left for future work.

7.4 Summary

This chapter discussed the implementation of the Pose Classifier component of the system developed for this project. **Section 7.1** discussed the three joint feature sets extracted from a joint set for pose recognition, namely the Position, Transformed and Angle features. **Section 7.2** discussed the implementation and training of the Pose Classifier using the Support Vector Machines described in **Chapter 6**.

Various test results were presented in **Section 7.3**. The results showed that the current system works well when tested against synthetic data. The results further showed that using Region Classifiers trained with larger data sets which are less constrained generally perform better than ones trained on more constrained data sets. Furthermore, the results showed that pose classifiers using the Mean-Shift based joint estimators generally perform better than the Centre of Gravity estimators, though there are instances when using Angle features, where the Centre of Gravity estimators might be preferred.

Chapter 8

Real World Testing

This chapter describes the set of tests performed to determine the real-world performance of the complete Pose Recognition system. The chapter first discusses the testing environment and then proceeds to show and discuss the obtained results.

8.1 Experimental Setup

The Microsoft Xbox Kinect depth sensor was used to obtain real world depth images. The stream of depth images was filtered using a temporal average filter, which calculated the average of the last 3 depth images retrieved from the Kinect. This smoothed the depth images considerably, though did cause a small amount of ghosting. The ghosting is however acceptable since we only measure stationary poses.

The hand coordinates retrieved from the OpenNI skeleton tracker was able to successfully find the hands, but the coordinates were not centred on the palm. We centred the coordinates by first cropping the 160x160 hand region from the depth image and then calculating the hand centre using a weighted centre of gravity algorithm, where the hand pixels closest to the centre of the image were given a larger weight. This weighting was used because the original hand coordinates were normally near the centre already and using a non-weighted algorithm might move the coordinates further away from the centre if the hand is incorrectly cropped.

Depth images of the right hand of 26 testing participants were captured using the Kinect camera, of which 20 participants' images were stored for testing and the other 6 for training. Each participant was seated 1 metre away from the camera and requested to show each of the 17 poses to the camera using their right hand. The system tracked their hands using the OpenNI skeleton tracker. A total of 30 sequential frames were captured for each pose shown by a participant, giving a total of 600 testing images for

each pose.

The hand regions of the depth images were classified using the RDF16_BL Region Classifier, which had the highest accuracy of the region classifiers. Joints were estimated from the resulting pixel probabilities using all three of the Joint Estimators discussed in **Chapter 5**. The various Joint Features were extracted from the joints sets and classified using the Pose Classifiers described in the previous chapter.

As mentioned above, a separate set of images was captured for training purposes. The training images were classified using the RDF16_BL Region Classifier and the resulting region map was used to extract the joints. These joints were used to train a new set of Pose Classifiers.

We found that libSVM was unable to train any of the Transform feature models when using the real world training data. Upon further investigation, it was found that the scaling factor calculated using the palm and second finger joints (*1) was not stable, causing large variations in the calculated features. We changed the Transform scaling factor to $\frac{1}{\text{image width}} = \frac{1}{160}$ for training and testing the real world data, which scales the feature values to the range $[0, 1]$. It does however indicate that a more robust scaling method is needed to make the algorithm invariant to changes in scale.

8.2 System Accuracy

The first set of tests measured the accuracy of the various Pose Classifiers, similar to the tests performed in **Chapter 7**. **Figure 8.2.1** shows the results of the Pose Classifiers trained using the synthetic data set. It can be seen that all of the classifiers perform poorly when tested against the real world test set.

There are two factors contributing to the poor results of these Pose Classifiers. Firstly, the noisy nature of the Kinect depth images results in the Region Classifier providing less accurate region maps for joint estimation. This can be especially troublesome when the depth images contain discontinuities on the hand surface.

Another factor contributing to the low performance is the cropping of the hand region. The synthetically generated depth images always cropped the wrist at a specific section. The cropped hand images from the real world images regularly contain large portions of the wrist and upper arm, which affect the positioning of the palm and wrist joints. This has a large effect on the extracted joint features, which in turn affect the final pose classification.

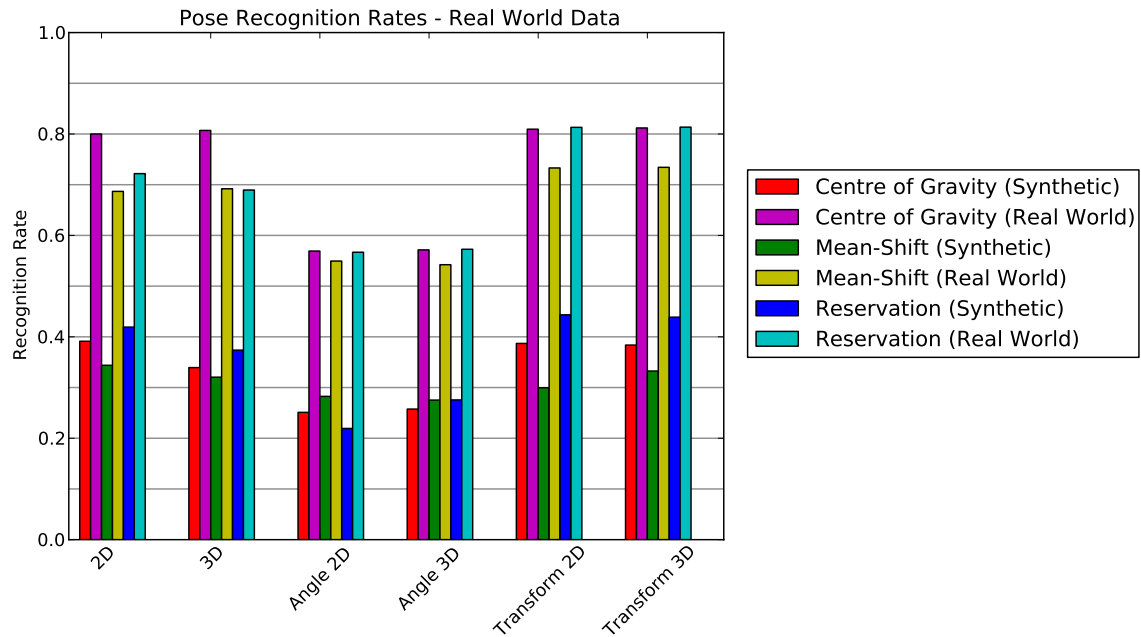


Figure 8.2.1: Real world recognition rate of the pose classifiers built using the RDF16_BL Region Classifier with various joint estimators and feature sets. The results include the recognition rate from pose classifiers trained using the synthetic data set and the real world data set.

As mentioned in the previous section, a separate set of Pose Classifiers was trained using real world training data in order to take the two factors mentioned above into account. The results of testing these Pose Classifiers on the real world testing set are shown in **Figure 8.2.1**. There is a considerable improvement in the performance of the various pose classifiers, since the classifier can model imperfections in the estimated joints caused by sensor noise. The performance can possibly be increased further by using more real world training data for the SVM models of the pose classifier.

The Mean-Shift-based pose classifier did not perform as well as expected, while the Centre of Gravity pose classifiers performed better than expected. The best performing classifier was the Reservation, Transform 3D classifier, which achieved a recognition rate of 81.35%. **Table B.4.1** and **Table B.4** show the numerical values of the other results.

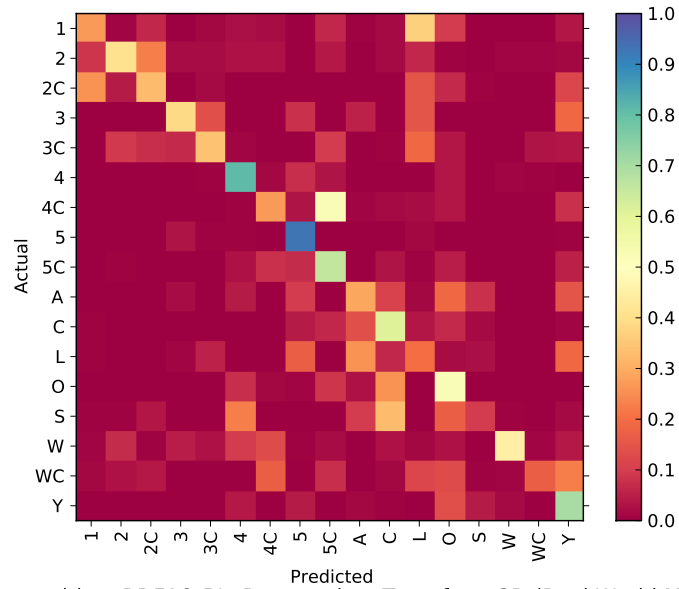
8.3 Confusion Properties

Confusion matrices were again generated from the pose classification results of the best performing classifier, namely the RDF16_BL region classifier with Transform 3D joint features extracted from the Reservation joint estimates. **Figure 8.3.1** shows the colour graphs corresponding to the confusion matrices. The top graph shows the results from

the SVM models trained using the synthetic data, while the bottom graph shows the results of the real world SVM models.

The top graph shows that the synthetic models are clearly unable to distinguish between any of the poses, given the faint diagonal line and general spread of values. The bottom graph shows that training the SVM models with real world data considerably increases the pose classifier's ability to distinguish between poses. However, similar poses are still confused, such as the ASL_C pose, which is frequently confused with the ASL_O and ASL_5C poses.

Pose Recognition: RDF16_BL, Reservation, Transform 3D (Synthetic Model)



Pose Recognition: RDF16_BL, Reservation, Transform 3D (Real World Model)

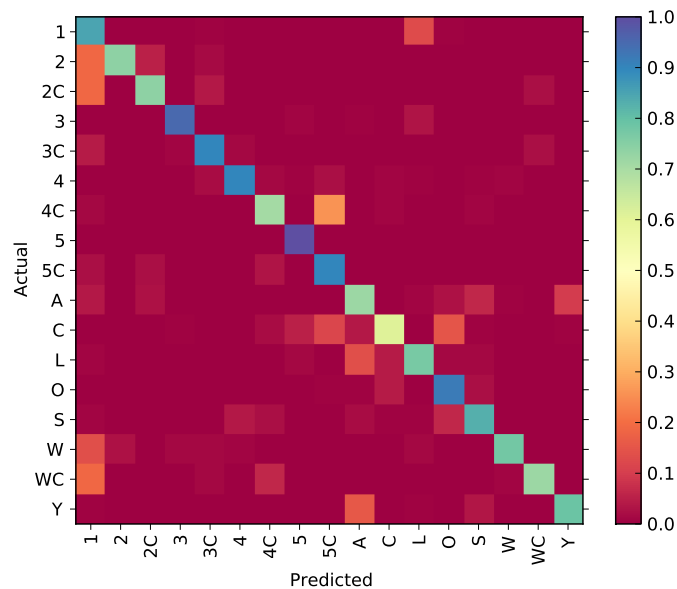


Figure 8.3.1: Confusion colour graphs indicating which poses the system confuses with which. The two graphs show the results when using the best performing pose classifier (RDF16_BL, Reservation, Transform3D). The top graph shows the results of the pose classifier using the SVM models trained with the synthetic data, while the bottom graph shows the results of the pose classifier trained using the real world data.

	1	2	2C	3	3C	4	4C	5	5C	A	C	L	O	S	W	WC	Y	
1	.27		.06		.01	.02	.02		.06		.02	.37	.1				.04	1
2	.09	.41	.23	.02	.02	.03	.03		.04		.02	.07				.01	.01	2
2C	.26	.04	.33		.02							.15	.07				.12	2C
3				.39	.14			.08		.06		.15					.19	3
3C		.1	.07	.07	.34	.01			.1			.19	.04			.03	.04	3C
4						.81	.01	.08	.04				.04		.01			4
4C							.27	.04	.52	.01	.02	.02	.04				.08	4C
5				.03				.93				.01						5
5C						.03	.08	.07	.66		.03		.05				.05	5C
A				.02		.04		.1		.29	.11	.01	.19	.08			.15	A
C								.04	.07	.14	.61	.04	.07	.02				C
L					.06			.17		.26	.06	.2	.02	.03			.19	L
O						.07	.01		.09	.03	.26		.52					O
S			.04			.23				.1	.33		.17	.1			.02	S
W		.07		.05	.03	.1	.13		.02		.03	.01	.03		.45	.01	.04	W
WC	.01	.03	.04				.17		.08		.01	.12	.13			.17	.23	WC
Y						.04		.04		.01			.14	.04	.02		.7	Y
	1	2	2C	3	3C	4	4C	5	5C	A	C	L	O	S	W	WC	Y	

Table 8.3.1: Confusion matrix showing the recognition rates of the different poses of the real world testing set using Region Classifier RDF16_BL, the Reservation Joint Estimator and the Transform 3D joint features, with SVM models trained using synthetic data. Empty entries indicate values smaller than 0.01.

	1	2	2C	3	3C	4	4C	5	5C	A	C	L	O	S	W	WC	Y	
1	.85											.13						1
2	.19	.74	.05		.02													2
2C	.19		.74		.04											.03		2C
3				.95								.03						3
3C	.04				.9	.01										.03		3C
4					.02	.9	.01		.02									4
4C	.01						.71		.26									4C
5								1.0										5
5C	.03		.03				.03		.9									5C
A	.04		.03							.72			.03	.06			.1	A
C							.02	.05	.12	.04	.61		.15					C
L	.01							.01		.14	.04	.77	.01	.01				L
O											.04		.92	.03				O
S						.04	.02			.02			.06	.83				S
W	.14	.03		.01	.01	.01						.01			.78			W
WC	.19				.01		.07									.72		WC
Y										.16				.04			.79	Y
	1	2	2C	3	3C	4	4C	5	5C	A	C	L	O	S	W	WC	Y	

Table 8.3.2: Confusion matrix showing the recognition rates of the different poses of the real world testing set using Region Classifier RDF16_BL, the Reservation Joint Estimator and the Transform 3D joint features, with SVM models trained using real world data. Empty entries indicate values smaller than 0.01.

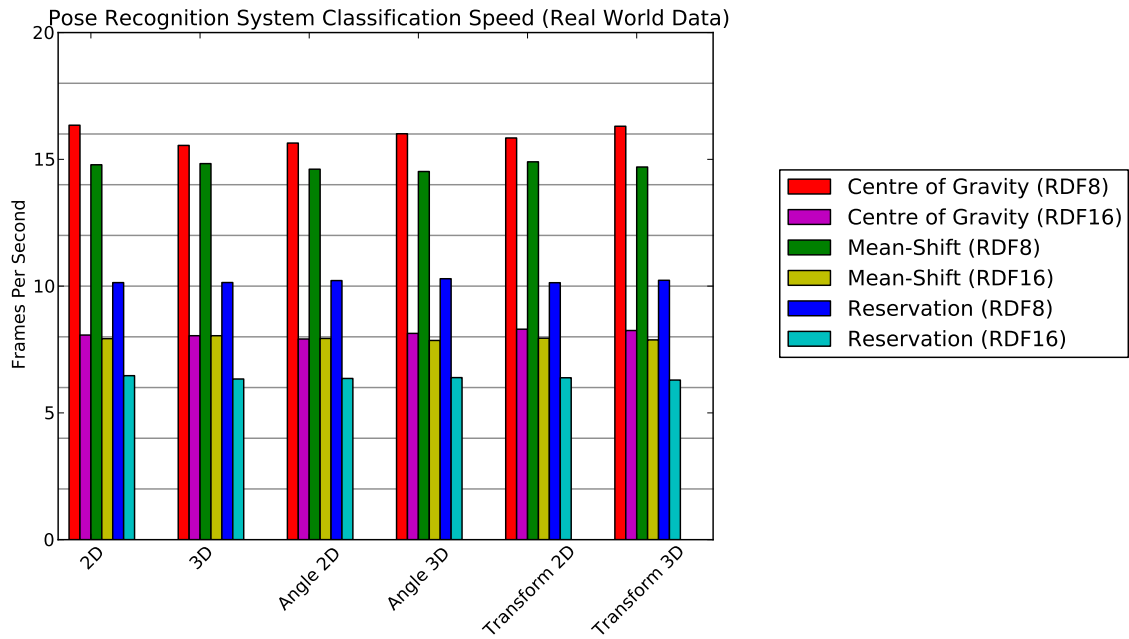


Figure 8.5.1: The classification speed of the complete Pose Recognition System.

8.4 McNemar Tests

McNemar tests were performed similar to those done for the synthetic data tests of **Chapter 7**, which are shown in **Figure 8.5.2**. The tests indicate that more testing data is needed to determine if the Reservation joint estimator is actually better than the Centre of Gravity estimator for real world data. Furthermore, we cannot confirm whether it is better to use 3D joint features as opposed to 2D features, given the high probability values of the corresponding McNemar tests.

8.5 System Classification Speed

The next set of tests measured the classification speed of the complete system when classifying the real world data. We used the RDF8_BL and RDF16_BL Region Classifiers combined with the various Joint Estimators, along with the Pose Classifiers trained using the real world training set. **Figure 8.5.1** shows the results of these tests.

The results show that the system is not able to perform in real-time when tested against the real world data, though it is slightly faster than the classification speed results of the synthetic data discussed in **Chapter 7**. This speed difference is likely attributed to small differences in the number of pixels needed to be classified between the two test sets and differences in the SVM models used.

	CP3D	CA2D	CA3D	CT2D	CT3D	MP2D	MP3D	MA2D	MA3D	MT2D	MT3D	RP2D	RP3D	RA2D	RA3D	RT2D	RT3D	
CP2D	.025			.002	~0											~0	~0	CP2D
CP3D				.483	.147											.117	.095	CP3D
CA2D			.142					~0	~0					.651	.492			CA2D
CA3D								~0	~0					.367	.815			CA3D
CT2D					.141											.348	.297	CT2D
CT3D																.755	.677	CT3D
MP2D							.164						.531					MP2D
MP3D												~0	.515					MP3D
MA2D									.001					~0	~0			MA2D
MA3D														~0	~0			MA3D
MT2D											.038	.012						MT2D
MT3D												.005						MT3D
RP2D																		RP2D
RP3D																		RP3D
RA2D															~0			RA2D
RA3D																		RA3D
RT2D																	.522	RT2D

Figure 8.5.2: McNemar tests to evaluate the comparisons of **Figure 8.2.1** (bottom).

8.6 Summary

This chapter discussed the setup and results of the real world tests performed on the system. It showed that the system can function as a real world pose classifier, though further testing is needed to determine the optimal parameters and components to use.

The chapter showed that the Region Classifier can be trained using synthetic data for the real world system, but the Pose Classifier should be trained using real world data. This is done to take into account the inaccuracies and noise found on the real world depth image retrieved from the Kinect depth sensor.

Chapter 9

Conclusion

This chapter serves as a summary of the various results and discussions presented in this thesis. The first section discusses the various results of the previous chapters. The second section discusses possible future work and improvements that can be made to the current system.

9.1 Project Discussion

This section briefly discusses the development and results of the complete Pose Classifier system. The section is divided into four subsections, which discuss the three system components, the Region Classifier, Joint Estimator and Pose Classifier as well as the results of the complete system.

9.1.1 Region Classifier

This project showed that it is feasible to train a Random Decision Forest-based Region Classifier using synthetic data. The resulting Region Classifier works well when used to classify depth images, but more importantly can be used as the basis for a Pose Recognition system for real world depth images, such as those retrieved from the Microsoft Kinect.

The various tests indicated that using Random Decision Trees trained from larger training sets work better than those trained from smaller sets. Furthermore, the tests showed that classifiers consisting of larger collections of RDTs within a forest deliver more accurate classification results at the expense of classification speed.

9.1.2 Joint Estimator

The project tested three different joint estimators, which were used to find the the 2D and 3D image coordinates of the various hand joints from the results obtained from the Region Classifier. The three estimators tested were the Centre of Gravity, Mean-Shift and Reservation estimators, where the latter estimator is our own modification of the Mean-Shift estimator which uses various heuristics to attempt to provide higher accuracy.

The various tests performed showed that the Mean-Shift and Reservation estimators proved to generally be the most accurate and consistent estimators with similar accuracy, performing better than the simple Centre of Gravity estimator. The Mean-Shift estimator is considerably faster than the Reservation estimator, though still slower than the simple Centre of Gravity estimator.

When using the Joint Estimators in a real world pose recognition system, it was found that the Centre of Gravity joints performed better than expected, while the Mean-Shift estimator performed worse than expected. The Reservation estimator performed well, but was not able to provide a considerable improvement in pose classification accuracy. McNemar tests further showed that more testing is needed to determine whether the Centre of Gravity joint estimator is better suited for a pose recognition system as opposed to a Reservation estimator.

9.1.3 Pose Classifier

The project used Support Vector Machines to successfully classify the pose from a set of estimated joints. Six different feature sets were extracted from the estimated joints, which include 2D and 3D Position, 2D and 3D Angle and 2D and 3D Transform features. The results from the synthetic and real world tests indicate that there is no significant difference in accuracy between the 2D and 3D feature variations, making the less complex 2D features more desirable. Furthermore, the accuracy of the Angle features in their current form were lacking compared to the Position and Transform features, though using a more complex set of angles might improve the performance.

The Transform features performed better than the Position features for the synthetic data, but required an adjustment to the scaling component of the feature extractor for the real world data. This was caused by less consistent joint estimation when estimating the joints from the real world images, which caused the scale factor to have a larger than expected range of values. A better scaling solution might be able to solve this problem.

9.1.4 Complete System

The complete system had a high accuracy when tested against the synthetic testing data, which contains none of the noise or artefacts found in the real world depth images retrieved from the Kinect depth sensor. It was able to classify the hand pose from depth images in real-time when using Region Classifiers consisting of eight RDTs or less. The highest achieved recognition rate for the synthetic tests was 98.72% when classifying Testing Set A with the RDF16_AL Region Classifier using Transform 3D features extracted from joints estimated using the Reservation joint estimator. When using Testing Set B, the largest recognition rate was 95.62%, which was achieved by the RDF16_BL Classifier using Transform 3D features extracted from joints estimated using the Reservation joint estimator.

The pose classifiers trained using the synthetic data did however struggle classifying the real world data, with the best performing classifier being the RDF16_BL Region Classifier using Transform 2D joint features and the Reservation joint estimator, with a recognition rate of 44.34%. The performance of the system was increased considerably by training the pose classifiers with real world data instead. The best real world classifier was the RDF16_BL Classifier using Transform 3D joint features and the Reservation joint estimator, with a recognition rate of 81.35%.

The system was however not able to perform classification of the depth images in real time. This was largely due to the complexity of the Region Classifiers, where the size of a Random Decision Forest affected the final classification time the most. This can be fixed by accelerating the region classification using a GPU, which would allow classifying multiple pixels simultaneously.

9.2 Similar Work

As discussed at the start of this thesis, our work is largely based on the work of Keskin et al. [7], applied the techniques for full body pose recognition described by Shotton et al. [2] on pose recognition of the human hand using depth images retrieved from the Kinect. Keskin's system achieved a final accuracy of 99.9% on real world data. Their test set consisted of 10 ASL digits from 10 different participants with a total of 20 000 images and was able to classify the images in real-time, though it is unclear whether different participants were used for training the classifiers.

Our system was tested using 17 different hand digits from 20 different individuals and achieved a recognition rate of 81.35%. The lower recognition score can be

attributed to the number and type of poses we used and also the number of participants used for testing and training. Our system confirmed that synthetic data can be used to successfully train a real-world Region Classifier as described by Shotton et al. [2], though for pose classification it is preferable to use real depth data.

This project further showed the influences of using different joint features when classifying the final hand pose. It showed that there is a negligible difference between using 2D joints as opposed to using 3D joints. It further showed that using the joint coordinates directly as features works well, though using features that are invariant to transformation, rotation and scaling can give a small increase in accuracy.

9.3 Future Work

There are numerous areas of the project which can be improved and extended. This section provides a few of the more important areas which should be pursued in the future.

9.3.1 Motion Recognition

The first logical extension of the system is adding a Motion Recognition component, which is able to classify the motion of the user's hands. We briefly discussed the use of a motion recognition component of a complete gesture recognition system in **Chapter 1**. The classified motion and pose can be combined to create a more descriptive hand gesture, extending the number of possible control signals the user can give the system as input.

9.3.2 Custom 3D Hand Modeller

As discussed in **Chapter 4**, we used Blender, a 3D modelling application, to generate the synthetic depth images and their corresponding labels. One of the limitations of using the application was the labelling of the depth images, which had to be done by hand. A better solution would be to create an application that can create depth images from a 3D model and also automatically label the surface pixels, based on the proximity of the hand joints. This would not only allow for a more standard way of labelling the hand regions, but also provides a means of assigning label probabilities to the pixels. The label probabilities would allow to better model the hand boundaries, as opposed to the hard decision boundaries used in our model.

9.3.3 Depth Sensors and Filters

As shown by the results, the system currently struggles when classifying hand poses from the depth images retrieved from the Microsoft Xbox Kinect. This could possibly be rectified by using more advanced depth image filters than the temporal average filter described in **Chapter 8**. These filters need to be able to smooth the surface of the hand, but should also be able to fill in discontinuities on the hand surface caused by shadowing artefacts.

Another solution is to simply use hardware that is able to more accurately extract the depth image from a scene. The newer version of the Kinect developed for Microsoft's Xbox One console should provide more accurate depth images, while other depth sensors such as those developed by Primesense (Carmines) and Asus (Xtion) can also be investigated.

Chapter 2 discussed why we did not perform camera calibration to remove camera distortion from the input depth images. We propose that future research should determine the difference in speed and accuracy between systems using camera calibration to remove distortion and systems that use the depth images directly.

9.3.4 Decision Forests

In **Chapter 3** we discussed the combination of multiple classifiers, in our case Decision Trees, to create a more accurate Region Classifier. We simply averaged the results of the individual classifiers, which proved to be fast and effective. We propose further investigation into using more complex techniques for combining these results, with the aim of improving the accuracy of the final Region Classifier.

9.3.5 Dot Products as Joint Features

In **Chapter 7** we discussed the various joint features investigated in this project. The angle features performed rather poorly, yet we do believe that there is potential in using the orientation of vectors connecting joints as features. We thus propose investigating the use of the dot product between vectors as a potential improvement over the angle features.

9.3.6 GPU Acceleration

The project can greatly benefit from GPU acceleration, especially for the Decision Tree classifiers. As the final results showed, the final system was not able to perform in

real-time when using 8-tree Region Classifiers, which influences the responsiveness of the system. The Region Classifiers could possibly be implemented on a GPU, given the simple depth features used. This would allow multiple pixels to be propagated down a Decision Tree simultaneously, which would greatly increase the performance of the classifier.

Bibliography

- [1] W. Freeman, D. Anderson, P. Beardsley, C. Dodge, M. Roth, C. Weissman, W. Yezauris, H. Kage, I. Kyuma, Y. Miyake *et al.*, "Computer vision for interactive computer graphics," *Computer Graphics and Applications, IEEE*, vol. 18, no. 3, pp. 42–53, 1998.
- [2] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 1297–1304.
- [3] H. Kim and A. Hilton, "Environment modelling using spherical stereo imaging," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, 27 2009-oct. 4 2009, pp. 1534 –1541.
- [4] S. H. Lee and S. Sharma, "Real-time disparity estimation algorithm for stereo camera systems," *Consumer Electronics, IEEE Transactions on*, vol. 57, no. 3, pp. 1018 –1026, August 2011.
- [5] J. Engel-Cox, R. Hoff, R. Rogers, F. Dimmick, A. Rush, J. Szykman, J. Al-Saadi, D. Chu, and E. Zell, "Integrating lidar and satellite optical depth with ambient monitoring for 3-dimensional particulate characterization," *Atmospheric Environment*, vol. 40, no. 40, pp. 8056–8067, 2006.
- [6] V. Verma, R. Kumar, and S. Hsu, "3D building detection and modeling from aerial lidar data," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2. Ieee, 2006, pp. 2213–2220.
- [7] C. Keskin, F. Kiraç, Y. Kara, and L. Akarun, "Real time hand pose estimation using depth sensors," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1228–1234.
- [8] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE, 2011, pp. 127–136.

- [9] L. Gallo, A. Placitelli, and M. Ciampi, "Controller-free exploration of medical image data: Experiencing the Kinect," in *Computer-Based Medical Systems (CBMS), 2011 24th International Symposium on*. IEEE, 2011, pp. 1–6.
- [10] E. Suma, B. Lange, A. Rizzo, D. Krum, and M. Bolas, "Faast: The flexible action and articulated skeleton toolkit," in *Virtual Reality Conference (VR), 2011 IEEE*. IEEE, 2011, pp. 247–248.
- [11] W. Fan, H. Wang, P. Yu, and S. Ma, "Is random model better? On its accuracy and efficiency," in *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, November 2003, pp. 51 – 58.
- [12] T. K. Ho, "Random decision forests," in *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, vol. 1. IEEE, 1995, pp. 278–282.
- [13] T. K. Ho, "The random subspace method for constructing decision forests," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 8, pp. 832–844, 1998.
- [14] R. Tsai, "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 4, pp. 323–344, 1987.
- [15] J. Heikkila and O. Silven, "A four-step camera calibration procedure with implicit image correction," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*. IEEE, 1997, pp. 1106–1112.
- [16] Z. Zhang, "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 11, pp. 1330–1334, 2000.
- [17] C. Zhang and Z. Zhang, "Calibration between depth and color sensors for commodity depth cameras," in *Multimedia and Expo (ICME), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–6.
- [18] D. Herrera C, J. Kannala, and J. Heikkilä, "Accurate and practical calibration of a depth and color camera pair," in *Computer Analysis of Images and Patterns*. Springer, 2011, pp. 437–445.
- [19] S. Mo, S. Cheng, and X. Xing, "Hand gesture segmentation based on improved Kalman filter and TSL skin color model," in *Multimedia Technology (ICMT), 2011 International Conference on*. IEEE, 2011, pp. 3543–3546.
- [20] Z. Ren, J. Yuan, and Z. Zhang, "Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera," in *Proceedings of the 19th ACM international conference on Multimedia*. ACM, 2011, pp. 1093–1096.

- [21] M. Elmezain, A. Al-Hamadi, and B. Michaelis, "A robust method for hand gesture segmentation and recognition using forward spotting scheme in conditional random fields," *2010 20th International Conference on Pattern Recognition*, no. 3, pp. 3850–3853, Aug. 2010. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5597659>
- [22] C. Joslin, A. El-Sawah, Q. Chen, and N. Georganas, "Dynamic gesture recognition," in *Instrumentation and Measurement Technology Conference, 2005. IMTC 2005. Proceedings of the IEEE*, vol. 3. IEEE, 2005, pp. 1706–1711.
- [23] C. Keskin, A. Erkan, and L. Akarun, "Real time hand tracking and 3D gesture recognition for interactive interfaces using HMM," in *In Proceedings of International Conference on Artificial Neural Networks*. Citeseer, 2003.
- [24] A. Malima, E. Ozgur, and M. Çetin, "A fast algorithm for vision-based hand gesture recognition for robot control," in *Signal Processing and Communications Applications, 2006 IEEE 14th*. IEEE, 2006, pp. 1–4.
- [25] M. Van Den Bergh, E. Koller-Meier, F. Bosche, and L. Van Gool, "Haarlet-based hand gesture recognition for 3D interaction," in *Applications of Computer Vision (WACV), 2009 Workshop on*. IEEE, 2009, pp. 1–8.
- [26] L. Bretzner, I. Laptev, and T. Lindeberg, "Hand gesture recognition using multi-scale colour features, hierarchical models and particle filtering," in *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*. IEEE, 2002, pp. 423–428.
- [27] D. Lowe, "Object recognition from local scale-invariant features," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 1999, pp. 1150–1157 vol.2.
- [28] O. Sushkov and C. Sammut, "Local image feature matching for object recognition," in *Control Automation Robotics Vision (ICARCV), 2010 11th International Conference on*, December 2010, pp. 1598–1604.
- [29] A. Barczak, F. Dadgostar, and C. Messom, "Real-time hand tracking based on non-invariant features," in *Instrumentation and Measurement Technology Conference, 2005. IMTC 2005. Proceedings of the IEEE*, vol. 3. IEEE, 2005, pp. 2192–2197.
- [30] A. Mittal, L. Zhao, and L. Davis, "Human body pose estimation using silhouette shape analysis," in *Proceedings. IEEE Conference on Advanced Video and Signal Based Surveillance, 2003.*, July 2003, pp. 263–270.

- [31] M. Hofmann and D. Gavrilu, "Multi-view 3D human pose estimation combining single-frame recovery, temporal integration and model adaptation," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 2214–2221.
- [32] T. Heap and D. Hogg, "Towards 3D hand tracking using a deformable model," in *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*. IEEE, 1996, pp. 140–145.
- [33] B. Stenger, P. Mendonca, and R. Cipolla, "Model-based 3d tracking of an articulated hand," in *Proceedings of the 20th IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'01)*, vol. 2. Institute of Electrical and Electronics Engineers, 2001, pp. 310–315.
- [34] N. K. I. Oikonomidis and A. Argyros, "Efficient model-based 3D tracking of hand articulations using Kinect," in *Proceedings of the 22nd British Machine Vision Conference, BMVC 2011, University of Dundee, UK*, 2011.
- [35] OpenNI, "The standard framework for 3d sensing," 2013. [Online]. Available: <http://www.openni.org>
- [36] avin2, "Sensorkinect," 2012, gitHub repository. [Online]. Available: <https://github.com/avin2/SensorKinect>
- [37] C. Conly, P. Doliotis, P. Jangyodsuk, R. Alonzo, and V. Athitsos, "Toward a 3D body part detection video dataset and hand tracking benchmark," in *Proceedings of the 6th International Conference on Pervasive Technologies Related to Assistive Environments*. ACM, 2013, p. 2.
- [38] joaquimrocha, "Skeltrack - a free software skeleton tracking library," 2013, gitHub repository. [Online]. Available: <https://github.com/joaquimrocha/Skeltrack>
- [39] V. Frati and D. Prattichizzo, "Using Kinect for hand tracking and rendering in wearable haptics," in *World Haptics Conference (WHC), 2011 IEEE*. IEEE, 2011, pp. 317–321.
- [40] M. N. Murty and V. S. Devi, *Pattern Recognition: An Algorithmic Approach*. University Press, Springer, 2011, pp. 123–146.
- [41] *Pattern Recognition*, edition=Fourth, pages=119–127, 198–203, 215–222, author=Theodoridis, S. and Koutroumbas, K., year=2009, publisher=Academic Press, Elsevier.
- [42] Y. Amit and D. Geman, "Shape quantization and recognition with randomized trees," *Neural Computation*, vol. 9, no. 7, pp. 1545–1588, 1997.

- [43] M. Šarić, “Libhand: A library for hand articulation,” 2011, version 0.9. [Online]. Available: <http://www.libhand.org/>
- [44] K. Fukunaga and L. Hostetler, “The estimation of the gradient of a density function, with applications in pattern recognition,” *Information Theory, IEEE Transactions on*, vol. 21, no. 1, pp. 32–40, 1975.
- [45] D. Comaniciu and P. Meer, “Mean shift: A robust approach toward feature space analysis,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 5, pp. 603–619, 2002.
- [46] T. Cacoullos, “Estimation of a multivariate density,” *Annals of the Institute of Statistical Mathematics*, vol. 18, no. 1, pp. 179–189, 1966.
- [47] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
- [48] C. J. Burges, “A tutorial on support vector machines for pattern recognition,” *Data Mining and Knowledge Discovery*, volume=2, number=2, pages=121–167, year=1998, publisher=Springer.
- [49] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, “A practical guide to support vector classification,” Department of Computer Science, National Taiwan University, 2010. [Online]. Available: <http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf>
- [50] A. Ben-Hur and J. Weston, “A user’s guide to support vector machines,” in *Data Mining Techniques for the Life Sciences*, pages=223–239, year=2010, publisher=Springer.
- [51] QuadProg++, “A c++ library for quadratic programming,” 2013, online. [Online]. Available: <http://quadprog.sourceforge.net>
- [52] CVXOPT, “Python software for convex optimization,” 2013, online. [Online]. Available: <http://cvxopt.org/>
- [53] C.-W. Hsu and C.-J. Lin, “A comparison of methods for multiclass support vector machines,” *Neural Networks, IEEE Transactions on*, vol. 13, no. 2, pp. 415–425, 2002.
- [54] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.

- [55] L. Gillick and S. J. Cox, "Some statistical issues in the comparison of speech recognition algorithms," in *Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 International Conference on.* IEEE, 1989, pp. 532–535.

Appendix A

American Sign Language Digit Set

A.1 Table of ASL Digits

Recognised American Sign Language Digits			
ASL Digit	Label	Variation	Variation Label
1 (One)	ASL-1	None	
2 (Two)	ASL-2	Closed	ASL-2C
3 (Three)	ASL-3	Closed	ASL-3C
4 (Four)	ASL-4	Closed	ASL-4C
5 (Five)	ASL-5	Closed	ASL-5C
A	ASL-A	None	
S	ASL-S	None	
L	ASL-L	None	
Y	ASL-Y	None	
W	ASL-W	Closed	ASL-WC
O	ASL-O	None	
C	ASL-C	None	

Table A.1.1: The list of 17 poses used to train and test our system, which includes 12 ASL digits of which 5 digits have two variations.

A.2 Tables of ASL Digit Images










Table of Gestures (Numbers)			
Pose	Image	Alternate View/Pose	Image
ASL-1			
ASL-2		ASL-2 (Closed)	
ASL-3		ASL-3 (Closed)	
ASL-4		ASL-4 (Closed)	
ASL-5		ASL-5 (Closed)	

Table A.2.1: The gesture set used to train the system (Numbers).











Table of Gestures (Letters)			
Pose	Image	Alternate View/Pose	Image
ASL-A			
ASL-S			
ASL-L			
ASL-Y			
ASL-W		ASL-W (Closed)	
ASL-O		(Right View)	
ASL-C		(Right View)	

Table A.2.2: The gesture set used to train the system (Letters).

Appendix B

Result Tables and Matrices

B.1 Region Classifier Results

Region Classifier Recognition Rate Comparison - Testing Set A					
Training Set (No. Training Images)	Decision Trees in Forest				
	1	2	4	8	16
Smallest (1.6k)	.4348	.4829	.5166	.5384	.5497
Smallest - Extended (1.6k)	.3831	.3936	.4766	.5344	.5751
Small (4.8k)	.5052	.5545	.5816	.5973	.6030
Medium (12.2k)	.562488	.5843	.6606	.6996	.7244
Large (24.4k)	.5918	.6247	.6878	.7251	.7453
Large - Extended (24.4k)	.5472	.5847	.6521	.6947	.7162

Table B.1.1: Table summarising the recognition rate of various Region Classifiers when tested against Testing Set A.

Region Classifier Recognition Rate Comparison - Testing Set A (Weighted)					
Training Set (No. Training Images)	Decision Trees in Forest				
	1	2	4	8	16
Smallest (1.6k)	.5972	.6462	.6766	.6915	.6972
Smallest - Extended (1.6k)	.5579	.5515	.6523	.7009	.7309
Small (4.8k)	.6420	.6892	.7131	.7218	.7257
Medium (12.2k)	.6866	.7021	.7621	.7890	.8059
Large (24.4k)	0.7017	.7301	.7789	.8045	.8169
Large - Extended (24.4k)	.6738	.7113	.7634	.7929	.8085

Table B.1.2: Table summarising the recognition rate of various Region Classifiers when tested against Testing Set A, weighted according to region representation in testing set.

Region Classifier Recognition Rate Comparison - Testing Set B					
Training Set (No. Training Images)	Decision Trees in Forest				
	1	2	4	8	16
Smallest (1.6k)	.3286	.3364	.3926	.4343	.4607
Smallest - Extended (1.6k)	.3404	.3491	.4185	.4716	.5082
Large (24.4k)	.4460	.4702	.5210	.5543	.5726
Large - Extended (24.4k)	.4919	.5264	.5905	.6321	.6539

Table B.1.3: Table summarising the recognition rate of various Region Classifiers when tested against Testing Set B.

Region Classifier Recognition Rate Comparison - Testing Set B (Weighted)					
Training Set (No. Training Images)	Decision Trees in Forest				
	1	2	4	8	16
Smallest (1.6k)	.4648	.4523	.5288	.5694	.5944
Smallest - Extended (1.6k)	.5018	.4937	.5858	.6302	.6573
Large (24.4k)	.5461	.5665	.6194	.6499	.6646
Large - Extended (24.4k)	.6145	.6473	.6979	.7262	.7412

Table B.1.4: Table summarising the recognition rate of various Region Classifiers when tested against Testing Set B, weighted according to region representation in testing set.

Region Classifier Classification Speed Comparison (FPS)					
Training Set (No. Training Images)	Decision Trees in Forest				
	1	2	4	8	16
Smallest (1.6k)	345.83	173.10	86.09	42.44	21.15
Small (4.8k)	318.22	156.13	78.18	38.99	19.71
Medium (12.2k)	308.62	151.22	76.50	38.40	19.03
Large (24.4k)	318.53	157.94	78.40	38.82	19.39

Table B.1.5: Table summarising the classification speed for various Region Classifiers.

	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR	
P0	.41	.24		.02	.04	.11														.15	.01	P0
P1	.02	.76	.07	.01		.07	.02													.02		P1
P2		.12	.71	.1			.05															P2
P3		.02	.06	.69				.04												.17		P3
R0		.01			.65	.15			.03	.04										.08		R0
R1		.03			.03	.8	.06			.04	.02											R1
R2		.02	.02			.09	.76	.04			.05											R2
R3				.03		.01	.08	.7				.07								.09	.01	R3
M0					.07	.01			.62	.12			.06	.02					.02	.04		M0
M1						.07			.03	.75	.08			.04								M1
M2						.02	.02			.05	.83	.04			.02							M2
M3								.04			.07	.76				.04				.06	.01	M3
I0									.02	.02			.75	.14				.03				I0
I1										.02			.03	.84	.08			.01				I1
I2											.03			.08	.77	.1						I2
I3												.03			.05	.85				.04		I3
T0		.01				.02	.02			.03	.03	.01			.01		.65	.18			.02	T0
T1																	.03	.81	.11		.01	T1
T2																		.03	.87	.03	.07	T2
PA				.02								.01				.02			.04	.79	.11	PA
WR																			.04	.07	.88	WR
	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR	

Table B.1.6: Confusion matrix showing the recognition rates of the different hand regions when testing RDF16_AL on Testing Set A. Empty entries indicate values smaller than 0.01.

	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR	
P0	.25	.21	.01	.02	.05	.16	.03										.01			.17	.07	P0
P1	.02	.55	.07	.02		.13	.04				.01									.05	.07	P1
P2		.13	.51	.11		.02	.09													.03	.08	P2
P3		.04	.07	.48		.02	.03	.04												.24	.07	P3
R0	.02	.03			.45	.16			.05	.06	.01									.13	.03	R0
R1		.06			.03	.62	.07			.08	.05									.03	.04	R1
R2		.03	.04	.01		.11	.57	.04			.1	.01								.01	.04	R2
R3				.04		.02	.09	.53			.01	.09								.15	.05	R3
M0					.1	.03			.44	.12	.01		.1	.03			.02	.02	.03	.06	.01	M0
M1					.02	.12			.03	.57	.09			.07				.02	.02	.01		M1
M2						.04	.06			.05	.65	.04		.02	.03	.02		.02	.02	.01	.01	M2
M3							.02	.06			.07	.59				.06			.01	.13	.03	M3
I0									.05	.04			.57	.15	.01	.01	.04	.04	.02	.02		I0
I1										.05	.03		.03	.68	.09			.06	.01			I1
I2										.01	.08			.1	.59	.12		.05	.03			I2
I3											.02	.05			.06	.69		.02	.07	.06	.02	I3
T0		.02				.03	.02			.03	.04	.01	.01	.01	.02	.01	.47	.2	.03	.02	.07	T0
T1											.01			.02	.01	.01	.03	.65	.15	.01	.06	T1
T2																		.04	.76	.04	.13	T2
PA				.02								.01				.02			.07	.67	.19	PA
WR																		.01	.1	.14	.73	WR
	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR	

Table B.1.7: Confusion matrix showing the recognition rates of the different hand regions when testing RDF16_AL on Testing Set B. Empty entries indicate values smaller than 0.01.

	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR	
P0	.3	.26		.02	.04	.12														.22		P0
P1	.02	.73	.08	.02		.07	.02													.04		P1
P2		.12	.69	.12			.05															P2
P3		.02	.06	.66				.04												.2		P3
R0		.01			.57	.18			.03	.04										.13		R0
R1		.03			.03	.79	.07			.03	.02									.01		R1
R2		.02	.03			.1	.74	.04			.06											R2
R3				.03		.01	.09	.66				.08								.11	.01	R3
M0					.07	.02			.56	.13		.01	.07	.02				.01	.03	.07		M0
M1						.08			.03	.72	.08			.04								M1
M2						.02	.02			.05	.82	.04			.02							M2
M3								.04			.08	.73				.05				.07	.01	M3
I0									.01	.02			.72	.15		.02		.03		.02		I0
I1										.02			.03	.83	.09			.01				I1
I2											.03			.1	.75	.11						I2
I3												.03			.06	.85				.04		I3
T0		.01				.02	.02			.03	.04	.02			.02	.01	.58	.21				T0
T1																.02	.03	.78	.14			T1
T2																		.02	.88	.04	.06	T2
PA				.01												.02			.04	.8	.11	PA
WR																			.04	.07	.88	WR
	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR	

Table B.1.8: Confusion matrix showing the recognition rates of the different hand regions when testing RDF16_BL on Testing Set A. Empty entries indicate values smaller than 0.01.

	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR	
P0	.24	.24		.02	.05	.15	.01													.22	.03	P0
P1	.02	.64	.08	.02		.11	.03													.05	.03	P1
P2		.13	.61	.13		.01	.07													.02	.03	P2
P3		.03	.07	.57			.01	.04												.24	.03	P3
R0		.01			.49	.18			.04	.05										.15	.02	R0
R1		.04			.03	.72	.07			.05	.03									.02	.01	R1
R2		.02	.04			.11	.67	.04			.08										.01	R2
R3				.03		.01	.1	.61				.08								.13	.02	R3
M0					.06	.02			.51	.13		.01	.08	.03				.01	.03	.08	.01	M0
M1					.01	.09			.03	.66	.1			.05				.01	.01			M1
M2						.03	.03			.05	.77	.04		.01	.03							M2
M3							.01	.05			.09	.68				.05				.09	.02	M3
I0									.02	.03			.67	.16		.02		.03	.01	.02		I0
I1										.03	.01		.03	.79	.09	.01		.02				I1
I2											.05			.1	.69	.12		.01				I2
I3											.01	.04			.06	.8			.01	.06	.02	I3
T0		.02				.03	.02			.02	.04	.02			.02	.01	.52	.22	.02		.03	T0
T1																.02	.03	.72	.16		.03	T1
T2																		.03	.82	.04	.11	T2
PA				.02												.02			.05	.74	.15	PA
WR																			.07	.1	.8	WR
	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR	

Table B.1.9: Confusion matrix showing the recognition rates of the different hand regions when testing RDF16_BL on Testing Set B. Empty entries indicate values smaller than 0.01.

B.2 Joint Estimator Results

Joint Estimator Speed Comparison - Smallest (FPS)					
Joint Estimator	Decision Trees in Forest				
	1	2	4	8	16
None	345.83	173.10	86.09	42.44	21.15
Centre of Gravity	331.11	168.94	84.58	42.04	20.94
Mean-Shift	213.42	126.90	71.20	38.15	19.85
Reservation	52.52	42.30	31.97	22.28	14.13

Table B.2.1: Table summarising the speed of the various Joint Estimators when using a Region Classifiers trained using the Smallest training subset.

Joint Estimator Speed Comparison - Large (FPS)					
Joint Estimator	Decision Trees in Forest				
	1	2	4	8	16
None	318.53	157.94	78.40	38.82	19.39
Centre of Gravity	305.62	155.24	77.36	38.41	19.17
Mean-Shift	188.81	113.13	64.21	34.61	18.16
Reservation	43.46	34.55	26.70	19.00	12.78

Table B.2.2: Table summarising the speed of the various Joint Estimators when using a Region Classifiers trained using the Large training subset.

Joint Estimator - Percentage of Joints Not Found																					
Joint Estimator (Region Classifier)	Joint Labels																				
	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR
Centre of Gravity (RDF16_BT)	64	.57	2.1	6.6	10	.01	.48	24	19	.08	0.0	9.9	10	.24	.24	1.9	16	.14	0.0	0.0	0.0
Centre of Gravity (RDF16_BL)	17	.08	1.8	6.7	1.4	0.0	0.61	22	5.5	.03	0.0	14	8.8	.75	.14	3.5	3.2	0.0	0.0	0.0	0.0
Mean-Shift (RDF16_BT)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Mean-Shift (RDF16_BL)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Reservation (RDF16_BT)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Reservation (RDF16_BL)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.2.3: Table summarising the percentage of joints not found for each joint estimator.

Joint Estimator - Percentage of Not Placed on Hand Surface																					
Joint Estimator (Region Classifier)	Joint Labels																				
	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR
Centre of Gravity (RDF16_BT)	65	2.4	8.9	8.3	15	2.0	5.1	24	28	4.2	2.3	9.9	24	5.7	2.7	2.2	22	5.4	.10	0.0	0.0
Centre of Gravity (RDF16_BL)	19	.82	6.5	8.6	3.6	.54	3.4	22	9.0	1.2	1.0	14	13	1.5	.56	3.6	3.8	.15	0.0	0.0	0.0
Mean-Shift (RDF16_BT)	2.2	2.0	2.6	.44	2.1	1.4	2.6	.03	6.6	3.3	1.9	.01	7.4	4.3	2.8	0.32	5.9	6.9	.26	0.0	0.0
Mean-Shift (RDF16_BL)	.92	.85	1.2	.04	.89	.45	1.4	.0	3.3	1.1	.61	0.0	3.7	1.6	.75	.18	4.4	2.1	0.0	0.0	0.0
Reservation (RDF16_BT)	1.9	.83	1.2	.30	1.2	.73	.95	.04	3.7	1.8	.65	.01	4.6	2.7	1.3	.16	2.5	1.7	.63	0.0	0.0
Reservation (RDF16_BL)	.86	.31	.88	.07	.53	.30	.61	0.0	1.8	.67	.29	0.0	2.3	.91	.30	.14	1.3	.26	0.0	0.0	0.0

Table B.2.4: Table summarising the percentage of joints not placed on the surface of the hand for each joint estimator.

Joint Estimator - Average Distance from Centre of Gravity Gold Standard (Pixels)																					
Joint Estimator (Region Classifier)	Joint Labels																				
	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR
Centre of Gravity (RDF16_BT)	9.9	2.3	2.4	3.4	4.0	2.4	2.2	3.2	4.3	2.2	2.1	2.8	4.1	2.3	2.1	2.2	4.7	3.0	2.5	1.9	2.1
Centre of Gravity (RDF16_BL)	2.9	1.1	1.6	1.9	1.3	1.3	1.5	2.2	1.7	1.2	1.4	2.2	1.7	1.2	1.3	1.4	1.5	1.2	.89	.83	.85
Mean-Shift (RDF16_BT)	4.1	2.8	2.7	3.5	3.7	2.7	2.6	3.5	3.9	2.3	2.5	3.0	3.8	2.4	2.3	2.4	3.9	3.3	2.7	1.9	2.2
Mean-Shift (RDF16_BL)	2.6	1.9	2.3	2.9	2.1	1.8	2.2	3.1	2.4	1.7	1.8	2.6	2.3	1.6	1.8	2.0	2.3	1.7	1.3	1.3	1.2
Reservation (RDF16_BT)	4.3	2.7	2.7	3.6	3.9	2.6	2.6	3.6	4.4	2.3	2.4	3.1	3.8	2.4	2.3	2.5	3.6	2.9	2.9	1.6	2.1
Reservation (RDF16_BL)	2.5	1.8	2.2	2.9	2.0	1.7	2.1	3.3	2.6	1.7	1.8	2.8	2.3	1.6	1.7	2.0	2.1	1.7	1.4	1.9	1.2

Table B.2.5: Table summarising the average distance from the Gold Standard each joint estimator placed joints, when testing against Testing Set A.

Joint Estimator - Standard Deviation from Centre of Gravity Gold Standard (Pixels)																					
Joint Estimator (Region Classifier)	Joint Labels																				
	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR
Centre of Gravity (RDF16_BT)	9.6	2.8	3.8	4.9	6.2	2.4	2.8	4.5	7.6	2.5	2.5	3.8	7.1	3.5	3.0	3.1	6.8	4.0	2.7	1.8	1.9
Centre of Gravity (RDF16_BL)	5.5	1.5	3.4	3.9	2.7	1.3	2.6	4.1	4.0	1.5	2.2	3.8	4.2	2.1	2.7	2.6	3.2	1.8	1.2	.74	.91
Mean-Shift (RDF16_BT)	4.8	2.8	3.3	4.5	5.7	2.5	2.5	3.9	6.8	2.4	2.4	3.4	6.1	3.1	2.8	3.1	6.1	4.5	2.8	1.6	2.0
Mean-Shift (RDF16_BL)	3.9	1.5	3.0	4.2	3.4	1.5	2.3	3.7	4.7	1.5	2.1	3.2	3.7	1.7	2.5	2.8	3.0	1.9	1.3	.73	.89
Reservation (RDF16_BT)	5.1	2.8	3.3	4.7	6.4	2.4	2.5	4.3	8.0	2.5	2.4	3.7	6.3	3.2	2.8	3.2	5.9	4.3	2.9	1.3	1.9
Reservation (RDF16_BL)	3.1	1.5	3.1	4.4	3.1	1.4	2.3	4.2	4.8	1.7	2.1	3.7	3.6	1.7	2.5	3.0	2.8	2.0	1.4	1.1	.88

Table B.2.6: Table summarising the standard deviation from the Gold Standard each joint estimator placed joints, when testing against Testing Set A.

Joint Estimator - Average Distance from Gold Standard of Testing Set A (Pixels)																					
Joint Estimator (Region Classifier)	Joint Labels																				
	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR
Centre of Gravity (RDF16_BT)	9.9	2.3	2.4	3.4	4.0	2.4	2.2	3.2	4.3	2.2	2.1	2.8	4.1	2.3	2.1	2.2	4.7	3.0	2.5	1.9	2.1
Centre of Gravity (RDF16_BL)	2.9	1.1	1.6	1.9	1.3	1.3	1.5	2.2	1.7	1.2	1.4	2.2	1.7	1.2	1.3	1.4	1.5	1.2	.89	.83	.85
Mean-Shift (RDF16_BT)	4.1	2.8	2.7	3.5	3.7	2.7	2.6	3.5	3.9	2.3	2.5	3.0	3.8	2.4	2.3	2.4	3.9	3.3	2.7	1.9	2.2
Mean-Shift (RDF16_BL)	2.6	1.9	2.3	2.9	2.1	1.8	2.2	3.1	2.4	1.7	1.8	2.6	2.3	1.6	1.8	2.0	2.3	1.7	1.3	1.3	1.2
Reservation (RDF16_BT)	4.3	2.7	2.7	3.6	3.9	2.6	2.6	3.6	4.4	2.3	2.4	3.1	3.8	2.4	2.3	2.5	3.6	2.9	2.9	1.6	2.1
Reservation (RDF16_BL)	2.5	1.8	2.2	2.9	2.0	1.7	2.1	3.3	2.6	1.7	1.8	2.8	2.3	1.6	1.7	2.0	2.1	1.7	1.4	1.9	1.2

Table B.2.7: Table summarising the average distance from the Gold Standard each joint estimator placed joints, when testing against Testing Set A.

Joint Estimator - Standard Deviation from Gold Standard of Testing Set A (Pixels)																					
Joint Estimator (Region Classifier)	Joint Labels																				
	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR
Centre of Gravity (RDF16_BT)	9.6	2.8	3.8	4.9	6.2	2.4	2.8	4.5	7.6	2.5	2.5	3.8	7.1	3.5	3.0	3.1	6.8	4.0	2.7	1.8	1.9
Centre of Gravity (RDF16_BL)	5.5	1.5	3.4	3.9	2.7	1.3	2.6	4.1	4.0	1.5	2.2	3.8	4.2	2.1	2.7	2.6	3.2	1.8	1.2	.74	.91
Mean-Shift (RDF16_BT)	4.8	2.8	3.3	4.5	5.7	2.5	2.5	3.9	6.8	2.4	2.4	3.4	6.1	3.1	2.8	3.1	6.1	4.5	2.8	1.6	2.0
Mean-Shift (RDF16_BL)	3.9	1.5	3.0	4.2	3.4	1.5	2.3	3.7	4.7	1.5	2.1	3.2	3.7	1.7	2.5	2.8	3.0	1.9	1.3	.73	.89
Reservation (RDF16_BT)	5.1	2.8	3.3	4.7	6.4	2.4	2.5	4.3	8.0	2.5	2.4	3.7	6.3	3.2	2.8	3.2	5.9	4.3	2.9	1.3	1.9
Reservation (RDF16_BL)	3.1	1.5	3.1	4.4	3.1	1.4	2.3	4.2	4.8	1.7	2.1	3.7	3.6	1.7	2.5	3.0	2.8	2.0	1.4	1.1	.88

Table B.2.8: Table summarising the standard deviation from the Gold Standard each joint estimator placed joints, when testing against Testing Set A.

Joint Estimator - Average Distance from Gold Standard of Shuffled Testing Set A using RDF16_BL (Pixels)																					
Joint Estimator (Region Classifier)	Joint Labels																				
	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR
Centre of Gravity	14	12	15	12	9.4	7.9	9.3	8.2	8.9	5.9	5.9	6.5	10	8.0	9.0	8.5	14	11	10	6.3	10
Mean-Shift	12	12	13	11	8.8	8.1	9.2	8.4	8.4	6.5	6.5	7.0	10	8.8	9.5	9.3	13	12	10	6.9	10
Reservation	12	11	13	11	9.8	7.6	8.9	8.1	9.3	6.3	6.2	6.4	10	8.6	9.2	8.9	13	12	10	8.0	10

Table B.2.9: Table summarising the average distance from the Gold Standard each joint estimator placed joints, when testing against Testing Set A with the shuffled algorithm applied to add noise similar to that of the Microsoft Kinect.

Joint Estimator - Standard Deviation from Gold Standard of Shuffled Testing Set A using RDF16_BL (Pixels)																					
Joint Estimator (Region Classifier)	Joint Labels																				
	P0	P1	P2	P3	R0	R1	R2	R3	M0	M1	M2	M3	I0	I1	I2	I3	T0	T1	T2	PA	WR
Centre of Gravity	9.1	7.4	8.0	7.1	6.4	5.5	6.1	5.6	7.1	3.9	4.1	4.1	8.1	5.5	5.8	4.9	9.9	7.6	5.4	3.1	3.4
Mean-Shift	8.3	7.2	7.7	6.6	6.4	5.4	5.7	4.9	6.9	4.0	3.8	3.8	7.4	5.5	5.5	4.7	10	7.3	5.4	2.9	3.3
Reservation	8.7	7.5	7.9	6.9	8.7	5.5	5.9	5.2	8.9	4.3	4.0	3.9	7.9	5.6	5.7	4.9	10	7.4	5.3	3.3	3.3

Table B.2.10: Table summarising the standard deviation from the Gold Standard each joint estimator placed joints, when testing against Testing Set A with the shuffled algorithm applied to add noise similar to that of the Microsoft Kinect.

B.3 Pose Classifier Results

Pose Classifier Recognition Rate Comparison - Testing Set A (Smallest)						
Joint Estimator (Region Classifier)	Joint Feature Set					
	Position 2D	Position 3D	Angle 2D	Angle 3D	Transform 2D	Transform 3D
Centre of Gravity (RDF16_AT)	.9215	.9229	.9334	.9333	.8421	.8422
Centre of Gravity (RDF16_BT)	.9482	.9454	.9574	.9547	.8819	.8817
Mean-Shift (RDF16_AT)	.9454	.9485	.9200	.9206	.9516	.9526
Mean-Shift (RDF16_BT)	.9531	.9644	.9258	.9282	.9634	.9638
Reservation (RDF16_AT)	.9464	.9571	.9254	.9258	.9545	.9549
Reservation (RDF16_BT)	.9542	.9670	.9316	.9315	.9626	.9639

Table B.3.1: Table summarising the recognition rate of various pose classifiers when tested on Testing Set A, using 16-tree region classifiers trained using the Smallest subset from both Training Set A and B.

Pose Classifier Recognition Rate Comparison - Testing Set A (Large)						
Joint Estimator (Region Classifier)	Joint Feature Set					
	Position 2D	Position 3D	Angle 2D	Angle 3D	Transform 2D	Transform 3D
Centre of Gravity (RDF16_AL)	.9774	.9682	.9283	.9279	.9701	.9729
Centre of Gravity (RDF16_BL)	.9757	.9706	.9201	.9191	.9767	.9728
Mean-Shift (RDF16_AL)	.9788	.9812	.8748	.8744	.9828	.9854
Mean-Shift (RDF16_BL)	.9794	.9814	.8634	.8642	.9822	.9847
Reservation (RDF16_AL)	.9810	.9842	.8852	.8852	.9853	.9871
Reservation (RDF16_BL)	.9812	.9847	.8764	.8761	.9850	.9863

Table B.3.2: Table summarising the recognition rate of various pose classifiers when tested on Testing Set A, using 16-tree region classifiers trained using the Large subset from both Training Set A and B.

Pose Classifier Recognition Rate Comparison - Testing Set B (Smallest)						
Joint Estimator (Region Classifier)	Joint Feature Set					
	Position 2D	Position 3D	Angle 2D	Angle 3D	Transform 2D	Transform 3D
Centre of Gravity (RDF16_AT)	.6923	.6940	.7102	.7103	.6365	.6366
Centre of Gravity (RDF16_BT)	.8519	.8531	.8797	.8780	.7774	.7742
Mean-Shift (RDF16_AT)	.6687	.6854	.6474	.6483	.6847	.6861
Mean-Shift (RDF16_BT)	.8789	.8925	.8404	.8437	.9021	.9001
Reservation (RDF16_AT)	.6899	.7150	.6690	.6696	.6992	.6989
Reservation (RDF16_BT)	.8836	.9052	.8502	.8511	.9036	.9039

Table B.3.3: Table summarising the recognition rate of various pose classifiers when tested on Testing Set B, using 16-tree region classifiers trained using the Smallest subset from both Training Set A and B.

Pose Classifier Recognition Rate Comparison - Testing Set B (Large)						
Joint Estimator (Region Classifier)	Joint Feature Set					
	Position 2D	Position 3D	Angle 2D	Angle 3D	Transform 2D	Transform 3D
Centre of Gravity (RDF16_AL)	.7710	.7630	.7007	.7001	.7730	.7660
Centre of Gravity (RDF16_BL)	.9185	.9113	.8522	.8531	.9274	.9122
Mean-Shift (RDF16_AL)	.7671	.7790	.6491	.6488	.7882	.7899
Mean-Shift (RDF16_BL)	.9353	.9335	.7890	.7883	.9498	.9532
Reservation (RDF16_AL)	.7767	.7907	.6467	.6649	.8003	.8012
Reservation (RDF16_BL)	.9400	.9445	.8059	.8068	.9542	.9561

Table B.3.4: Table summarising the recognition rate of various pose classifiers when tested on Testing Set B, using 16-tree region classifiers trained using the Large subset from both Training Set A and B.

Pose Classifier Classification Speed Comparison - Synthetic Data (FPS)						
Joint Estimator (Region Classifier)	Joint Feature Set					
	Position 2D	Position 3D	Angle 2D	Angle 3D	Transform 2D	Transform 3D
Centre of Gravity (RDF8_BL)	14.73	14.65	14.65	14.66	14.77	14.67
Centre of Gravity (RDF16_BL)	7.22	7.11	7.22	7.29	7.28	7.29
Mean-Shift (RDF8_BL)	13.60	13.47	13.39	13.39	13.60	13.69
Mean-Shift (RDF16_BL)	7.00	6.97	6.95	6.99	7.02	7.04
Reservation (RDF8_BL)	9.91	9.84	9.78	9.82	9.89	9.91
Reservation (RDF16_BL)	5.73	5.70	5.73	5.77	5.80	5.77

Table B.3.5: Table summarising the classification speed of various pose classifiers when tested on Testing Set A, using 8- and 16-tree region classifiers trained using the Large subset of Training Set B.

	1	2	2C	3	3C	4	4C	5	5C	A	C	L	O	S	W	WC	Y	
1	.99																	1
2		.95	.05															2
2C			.99															2C
3				.98	.02													3
3C				.02	.98													3C
4						1.0												4
4C							.99											4C
5								1.0										5
5C									1.0									5C
A										.99								A
C											.99							C
L												1.0						L
O													.97	.02				O
S													.01	.98				S
W															.99			W
WC							.03									.97		WC
Y																	1.0	Y
	1	2	2C	3	3C	4	4C	5	5C	A	C	L	O	S	W	WC	Y	

Table B.3.6: Confusion matrix showing the recognition rates of the different poses of Testing Set A using Region Classifier RDF16_BL, the Reserovation Joint Estimator and the Transform 3D joint features. Empty entries indicate values smaller than 0.01.

	1	2	2C	3	3C	4	4C	5	5C	A	C	L	O	S	W	WC	Y	
1	.97		.02															1
2		.94	.05															2
2C	.01	.03	.94															2C
3				.92	.07													3
3C				.06	.94													3C
4						.98									.01			4
4C							.95		.02							.01		4C
5								.99										5
5C							.02		.96									5C
A										.98								A
C									.01		.97							C
L												.98						L
O													.94	.04				O
S													.03	.96				S
W						.02									.95	.01		W
WC							.04								.01	.92		WC
Y										.02							.98	Y
	1	2	2C	3	3C	4	4C	5	5C	A	C	L	O	S	W	WC	Y	

Table B.3.7: Confusion matrix showing the recognition rates of the different poses of Testing Set B using Region Classifier RDF16_BL, the Reservoir Joint Estimator and the Transform 3D joint features. Empty entries indicate values smaller than 0.01.

B.4 Real World Results

Pose Classifier Recognition Rate Comparison - Real World						
Joint Estimator (SVM Model)	Joint Feature Set					
	Position 2D	Position 3D	Angle 2D	Angle 3D	Transform 2D	Transform 3D
Centre of Gravity (Synthetic)	.3912	.3393	.2510	.2575	.3869	.3837
Centre of Gravity (Real World)	.7999	.8070	.5691	.5714	.8094	.8118
Mean-Shift (Synthetic)	.3438	.3204	.2825	.2752	.2995	.3325
Mean-Shift (Real World)	.6868	.6919	.5494	.5422	.7330	.7342
Reservation (Synthetic)	.4192	.3737	.2193	.2755	.4434	.4387
Reservation (Real World)	.7218	.6894	.5668	.5726	.8131	.8135

Table B.4.1: Table summarising the recognition rate of various pose classifiers when tested on real world data, using RDF16_BL as the region classifier and pose classifiers trained using synthetic and real world data.

Pose Classifier Classification Speed Comparison - Real World (FPS)						
Joint Estimator (Region Classifier)	Joint Feature Set					
	Position 2D	Position 3D	Angle 2D	Angle 3D	Transform 2D	Transform 3D
Centre of Gravity (RDF8_BL)	16.34	15.54	15.64	16.00	15.84	16.30
Centre of Gravity (RDF16_BL)	8.07	8.04	7.91	8.13	8.300	8.24
Mean-Shift (RDF8_BL)	14.78	14.83	14.61	14.52	14.90	14.69
Mean-Shift (RDF16_BL)	7.92	8.04	7.93	7.85	7.94	7.87
Reservation (RDF8_BL)	10.13	10.14	10.21	10.29	10.13	10.22
Reservation (RDF16_BL)	6.46	6.33	6.35	6.39	6.38	6.29

Table B.4.2: Table summarising the classification speed of various pose classifiers when tested on real world testing data, using 8- and 16-tree region classifiers trained using the Large subset of Training Set B and pose classifiers trained using the real world training data.